

ROGRAMADORES

Revista especializada para usuarios de PC

AÑO 2. Nº 14
1250 PTAS.
ARGENTINA 9'50 $
CHILE 3000 $

Y además:
- Programación bajo Windows
- Las shells de Unix
- Curso de Clipper
- Optimización de ensamblador
- Programación C++

Reportaje Assembly 95

Programación de la Sound Blaster

Velocis 1.2.1

Hipertexto: el sistema de ayuda de Windows 3.1

TORNEO DE PROGRAMACIÓN DE DEMOS
250.000 Ptas. en efectivo para el ganador

Curso de Ray Tracing

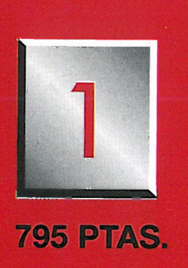




CURSO

AutoCAD

DE DISEÑO TÉCNICO



795 PTAS.

INTRODUCCIÓN AL CURSO

CAD una forma distinta de dibujar

¿Qué herramientas vamos a usar?

ESTRUCTURACIÓN DEL CURSO

Pautas y directrices

REQUERIMIENTOS TÉCNICOS

¿Qué se necesita para seguir este curso?

CURSO AutoCAD DE DISEÑO TÉCNICO

ESTE CD-ROM SE ENTREGA CONJUNTA E INSEPARABLEMENTE CON EL CURSO AutoCAD DE DISEÑO TÉCNICO

© TOWER COMMUNICATIONS © Autodesk Development BV

ESTE CD-ROM CONTIENE LA VERSIÓN DE AutoCAD LT 2 NECESARIA PARA SEGUIR ESTE CURSO, ADEMÁS DE UN GRUPO DE UTILIDADES Y FICHEROS. TIF Y DWG TAMBIÉN CONTIENE LOS FICHEROS DE PRÁCTICAS DEL FASCÍCULO Nº 2

TOWER COMMUNICATIONS

¡GRATIS! CD-ROM con la versión de AUTOCAD LT2 necesaria para seguir este curso

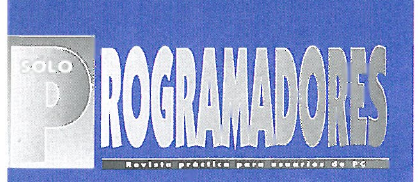
YA A LA VENTA EN TU QUIOSCO

SOLICITA INFORMACIÓN llamando al teléfono (91) 741 26 62 de 9 a 14 y de 15:30 a 18:3 o por Fax al (91) 320 60 72





PCP 4



CARTA DEL DIRECTOR



ATRAPADOS EN LA RED

Cuando yo empecé en esto de la programación, las redes eran los aparejos de pesca de los hombres del mar. Seguro que había alguna funcionando pero a mí, como al 90% de los informáticos de la época, nos pasaba absolutamente inadvertida. Hoy es totalmente diferente. Si no estás conectado no eres nada. Compuserve, Internet, Fidonet, hay que *engancharse* a algo por cable. Ahí está la información.

Hace menos de un mes, el diario EL MUNDO puso su WEB en marcha y en menos de una semana recibió la visita de 1.500 navegantes ansiosos de actualidad. Del mismo modo, ID Software (los autores del archiconocido DOOM) incluyeron en su servidor el pasado 10 de agosto las primeras pantallas de QUAKE (la continuación de DOOM) y en poco menos de una semana se estaban publicando en las revistas de todos los rincones del planeta.

Las redes son hoy en día la forma más barata de obtener y divulgar información y, por ahora, el tráfico es libre. Algunos sectores conservadores del panorama político norteamericano han tratado de *ganar puntos* ante sus electores proclamando a voces que esa *guarrada* de Internet, por la que circulan miles de GIFs y TIFs de señoritas semi desnudas, debe ser controlada y sometida a censura. No os podéis ni imaginar la que se puede *liar* como la iniciativa siga adelante. Para empezar, el flujo de información de Internet es, por la morfología de la red, totalmente incontrolado: los servidores de archivos no verifican la información y los que actúan de repetidores no pasan filtro alguno a los ficheros que re-transmiten. Imaginaos tener que reconocer imágenes eróticas en cada servidor con los cientos de *Gigas* que se mueven por la red cada día. A buen seguro que el genial creador de la propuesta desconocía por completo el *modus operandi* de Internet.

¿Es rentable pagar por tener acceso a Internet? La respuesta es: para todos los profesionales SI. ¿Por qué para los profesionales y no para los aficionados? Las personas que vivimos de la informática podemos encontrar en Internet información o rutinas sobre un tema determinado que nos ahorren semanas de trabajo. Muchas veces una idea es la clave para resolver un complicado problema y esa idea, seguro que se halla en algún lugar de los múltiples forums de programación de Internet.

Al principio, *the english language* es una barrera para navegar libremente pero, en cuestión de semanas la red es una forma más de practicar con soltura el idioma de Shakespeare.

Las redes se muestran como un instrumento que cambiará la forma de vida y de trabajo de toda la sociedad en el próximo siglo. Cientos de empleos administrativos, contables y asesorías se podrán realizar vía telemática cuando los módems duales (voz+canal de datos) se impongan y sean mayoría. Tal vez de ahí, del ahorro en los desplazamientos salga esa hora diaria que a mí me falta todos los días para pasar con mi familia y amigos. Ojalá.

MARIO DE LUIS



OCTUBRE 1995. Número 14
Es una publicación de



COMMUNICATIONS, S.R.L.

Editor
Antonio M. Ferrer Abello
Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador técnico
Eduardo Toribio Pazos

••••••

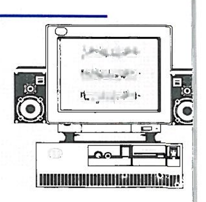
Directora Comercial
Carmina Ferrer
Director de Producción
Carlos Peropadre
Publicidad
Magdalena Pedreño Llorente

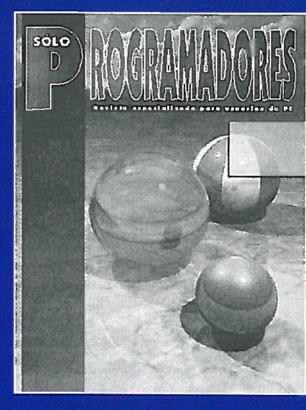
Colaboradores
Fernando de la Villa, Fernando J. Echevarrieta, Juan M. Martín, Luis Martín, Jose María Peco, José Carlos Remiro, Luis Crespo, Enrique de Alarcon, Jorge R. Regidor, Agustín Guillen, Juan Ramon Lehmann, Hector Martínez, Alvaro Silgado, David Aparicio, Ignacio Cea, Juan José Samper, Enrique Castañon

Edición técnica
Emilio Castellano
Maquetación
Fernando García Santamaría
Tratamiento de imagen
Josefa Fernández Martínez
Servicios Informaticos
Digital Dreams Multimedia S.
Ilustraciones
Miguel Aicon
Secretaría de Redaccion
Rosa Arroyo
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Marqués de Portugalete, 10
28027 MADRID
Telf.: 741 26 62 / Fax: 320 60 72
Filmación
Duvial
Impresión
G.D.B.
Distribución
MIDESA

Depósito legal: M-26827-1994
ISSN: 1134-4792

SUMARIO





Portada cortesía de Álvaro Silgado

SUMARIO








1er Torneo de Demos e Infografía

El torneo sigue en marcha, en la redacción se arma un revuelo cada vez que llega una nueva demo. Sólo Programadores os anima a concursar. Los que nunca os hayáis presentado a un concurso esta es vuestra oportunidad, y los más veteranos a intentar superarse. Os recordamos que se trata de un campeonato de **Demos** e **Infografía**. Dentro del apartado de demos se incluye también el de **Intros** (como las demos pero más pequeñas). Además del apartado de Infografía se encuentran el de **Vídeo** y el de **Morphing**, de los cuales se habla más adelante.

Existe un suculento premio de **250.000 pesetas en metálico** para los ganadores además de numerosos paquetes de software.

El torneo abrió sus puertas a los más intrépidos el pasado mes de agosto. Durante varios números recogeremos los trabajos realizados por todos aquellos que se animen a participar. El plazo de entrega concluirá el 15 de Diciembre.

BASES DEL CONCURSO

- Para concursar es imprescindible rellenar y enviar la tarjeta de participación que se adjunta en estas páginas.
- El jurado estará compuesto por la redacción de Sólo Programadores.
- Se podrán enviar uno o varios trabajos en cualquiera de las categorías, que son las siguientes:

- DEMOS:

- No entrarán en el concurso aquellas demos que hayan obtenido primeros puestos en campeonatos internacionales. Sin embargo se podrán enviar para incluirlas en el CD-ROM en el que se publiquen los resultados del concurso.
- Existen varias ramificaciones en la categoría de demos:

INTROS

Son demos de longitud mucho más reducida. Habrá dos grupos: de tamaño menor o igual a **4 Kb** (4.096 bytes) y hasta **64 Kb** (65.536 bytes). Tradicionalmente las primeras no suelen incluir sonido, mientras que las segundas suelen aportar alguna melodía o efecto sonoro.

Las longitudes se refieren al tamaño del fichero ejecutable (que vemos desde el DOS). El archivo .EXE o .COM puede estar empaquetado con la utilidad PK-lite. Dicha utilidad comprime un archivo EXE y genera otro EXE de menor tamaño. Este último cuando se ejecuta se descomprime en RAM en tiempo real quedándose con su tamaño original que se supone es superior. Así Intros que ocupan 100 Kb se reducen a 60 Kb con la utilidad PK-Lite o similar, y pueden entrar en la categoría de 64 Kb.

DEMOS

Se entiende por longitud de la demo la suma de las longitudes de los ficheros necesarios para la ejecución de la demo, bien sea un sólo archivo EXE o uno ejecutable más otros de datos. Dichos ficheros también pueden utilizar técnicas de autodescompresión en memoria como las comentadas en el apartado anterior.

Las demos tendrán un tamaño superior a 64 Kb. La única limitación es la de 4 Mb de tamaño máximo de archivos. Se pueden programar demos con extensores de DOS como el DOS4GW de Watcom puesto que el ordenador de prueba será un **486 DX2-66**. Dichos extensores deberán ser enviados en el disquete para poder evaluar la demo.

No se contabilizará como válida la longitud del archivo si está comprimida con uno de los compresores habituales (ARJ, LZH, ZIP, etc.).

- Se enviarán dos copias de la demo. Esto se puede hacer de varias formas. Una consiste en copiar el archivo de la demo dos veces en el disquete (con distinto nombre). Otra forma es enviar dos discos con una copia del archivo en cada uno.

Si es necesario la demo estará comprimida, y si supera la cantidad de 1,4 Mb se podrá empaquetar por volúmenes. **Sólo se admitirán las siguientes compresiones: ARJ, ZIP y LZH.**

En los disquetes irá una pegatina como la que aparece en la **figura 2**. En primer lugar irá el "handle" o nombre del participante/grupo para el concurso y en las casillas se pondrá una X indicando las categorías a las que pertenecen los trabajos enviados en dichos disquetes. Por ejemplo, si un grupo de amigos envía una Intro de 4 Kb en un disquete, pondrá en la pegatina el nombre del grupo de programación y una X en DEMOS, porque la categoría INTROS pertenece a DEMOS.

- INFOGRAFIA:

En las dos categorías, **Imagen** y **Animación**, se admitirán solamente trabajos realizados con programas de infografía o

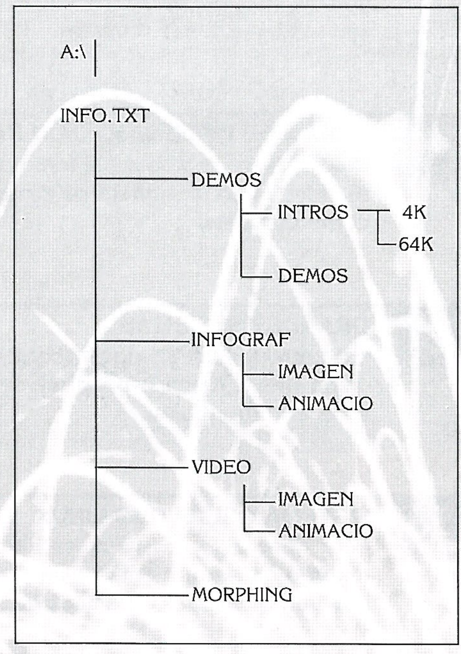


Figura 1.

imagen sintética como pueden ser POV, Topas, 3D Studio, etc.).

- **Imagen.** Aquí habrá sub-categorías para englobar los trabajos en función de la resolución y número de colores.
  Sólo serán válidos los trabajos en formato TIF, GIF, JPG y TGA.
  Además de incluir la infografía final se adjuntará obligatoriamente al menos una pantalla del trabajo en fase de construcción. Esto se hará para certificar que el trabajo está siendo realizado por su correspondiente autor.
- **Animación.** En el apartado de infografía en movimiento los trabajos serán admitidos en cualquier formato siempre y cuando se incluya el programa (DOS o Windows solamente) que visualize la animación en un PC 486-DX con 8 Mb RAM y tarjeta True Color VESA.

- VIDEO:

Aquí hay dos grupos similares a la categoría anterior: **Imagen** y **Animación.**

- **Imagen.** Es para los trabajos o composiciones fotográficas en los que se hallan empleado programas específicos para el fotomontaje (Photoshop, Picture Publisher, Photostyler, etc.). Se admitirán sólamente los formatos TIF, GIF, JPG y TGA.

(HANDLE O NOMBRE DE PARTICIPACIÓN)	
DEMOS [ ]	INFOGRAFIA [ ]
VIDEO [ ]	MORPHING [ ]

Figura 2.

- Animación. Aquí se engloban los trabajos que hayan sido digitalizados como vídeos caseros de humor o imágenes con cierto tratamiento digital (efectos de cortina, fades, etc.). Se admitirán solamente trabajos en formato AVI (Video for Windows) y MOV (QuickTime).

- MORPHING:

Se entiende por trabajos de morphing aquellos en los que a través de una secuencia de vídeo se observan transformaciones de objetos, animales o personas. Para estas secuencias no se permite utilizar personajes del ámbito público (políticos, actores, etc.). Sólo se admitirán ficheros AVI, MOV, FLI, FLC y programas que hagan Morphing en tiempo real siempre y cuando hayan sido programados con rutinas propias de los concursantes.

SOBRE EL MATERIAL ENVIADO

Sólo se admitirá el soporte de disquete y cintas de streamer compatibles QIC-80 para enviar los trabajos. Si se desea que se devuelvan los disquetes y las cintas de streamer hay que adjuntar los correspondientes sobres ya franqueados para que Sólo Programadores realice la devolución.

Los disquetes o las cintas donde se envíen los trabajos tendrán una estructura de directorios como la que se muestra en la figura 1.

En cada subdirectorio se colocarán el/los programa/s que vayan a participar. En el fichero INFO.TXT se escribirá en formato ASCII el nombre y teléfono de contacto del representante, así como una breve descripción de cada uno de los trabajos que se envíen.

LOS PREMIOS

Se le darán 250.000 pesetas en metálico a la mejor y más espectacular Demo o Infografía. Los finalistas de las otras categorías recibirán numerosos programas de regalo.

TARJETA DE PARTICIPACIÓN

Nombre:...Apellidos:..Edad: ....

Domicilio:...Cód.postal:.............................Teléfono:...

Pseudónimo o handle para el concurso: ....

CATEGORÍA DE PARTICIPACIÓN

(En cada modalidad se pondrá el número y los nombres de los trabajos enviados)

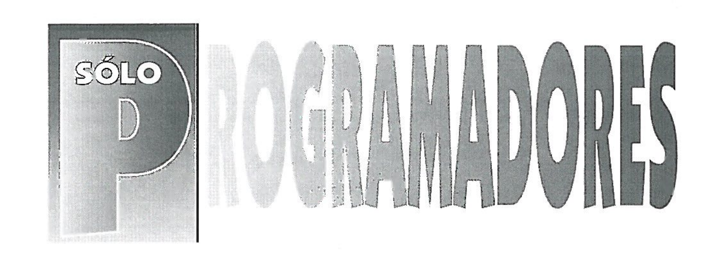
- Demos [ ]: ____________________
- Infografía (ficheros .GIF, .TGA, .TIF, FLI, FLC, etc.)
  - Imagen fija [ ]:____________________
  - Animaciones [ ]: ____________________
- Vídeo digital (ficheros .GIF, .TGA, .TIF, .AVI, .MOV, etc.)
  - Imagen fija [ ]: ____________________
  - Animaciones [ ]: ____________________
- Morphing [ ]: ____________________

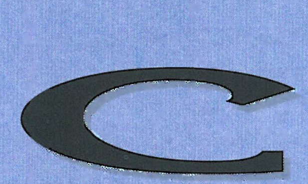
NORMATIVA PARA EL CONCURSO

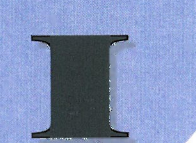
- Los trabajos enviados podrán estar realizados por una o varias personas. Si se trata de varias personas solamente el representante aparecerá en la tarjeta de participación. Los datos personales de los otros componentes del grupo serán adjuntados en un folio aparte escritos a máquina.
- No es necesario incluir los fuentes de los programas enviados.
- La responsabilidad de los trabajos enviados atañe tanto al representante que aparece en la tarjeta de participación como a los otros componentes del grupo si es que los hay. Ambos tendrían que responder ante la ley si terceras personas reclamasen la autoría del material o trabajos enviados para este concurso.
- La propiedad intelectual y los derechos de explotación del material enviado pertenecen tanto al representante del grupo como a los demás componentes si los hay. Con la participación en este concurso se concede a la editorial Tower Communications el permiso para publicar en CD-ROM el software enviado para dicho concurso.
- La redacción de Sólo Programadores se reserva el derecho de descalificar a algún participante por un mal funcionamiento del programa enviado o porque dicho programa contenga algún virus informático. También quedarán descalificados aquellos trabajos cuyo contenido sea de mal gusto.
- Enviar esta tarjeta de participación implica aceptar las bases del concurso y la normativa que se estipula en estos apartados.
- La participación en el concurso no será válida si falta algún dato, si hay información incorrecta o bien si la tarjeta de participación es una fotocopia y no la original que viene en la revista. Tampoco será válida la participación en el concurso si la tarjeta enviada no está firmada por el representante.
- La revista Sólo Programadores se reserva el derecho de modificar o ampliar las bases del concurso sobre la marcha si fuese necesario.

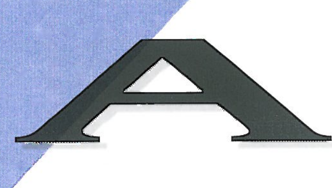
Acepto, en representación propia o del grupo, las bases del concurso que aparecen en el número 12 de Sólo Programadores.

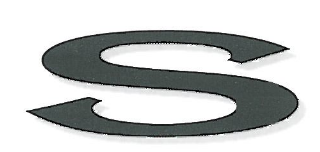
Firma del representante:











ACUERDO ENTRE CHIPCOM Y HEWLETT-PACKARD

Chipcom Corporation ha anunciado la reciente firma de un acuerdo estratégico de alianza corporativa a nivel mundial con la organización de servicio y soporte de la compañía Hewlett-Packard. Bajo la denominación Worldwide Platinum Support del acuerdo, Hewlett-Packard se ha convertido en el primer proveedor de servicios Chipcom a nivel mundial.

Según los términos de este acuerdo, Hewlett-Packard tiene autorización para vender, dar servicio y soporte a todos los productos hardware y software de sistemas de comunicación inteligentes de Chipcom, entre los que se incluyen los siguientes: el conmutador de red Turbo StarBridge, el concentrado de Sistema Online, el sistema de conmutación Oncore, el *hub* de conmutación de Red Galáctica, el software de gestión de redes Ondemand NCS junto a las aplicaciones relacionadas con el mismo, y las familias de productos Onsemble StackSystem.

Para más información:
Chipcom
Philippe Bernard
Tel: (33-1) 39 10 14 70

HITACHI LICENCIA LA TECNOLOGÍA COMPCORE MPEG-2

Hitachi Europe Ltd. ha anunciado la integración del motor MPEG-2 de CompCore Multimedia como base para el desarrollo de circuitos integrados decodificadores. Ambas compañías están desarrollando bajo un acuerdo tecnológico los productos, estando previsto los primeros a nivel de muestra a principios del año 1996.

Los nuevos productos MPEG-2 están basados en los decodificadores ya existentes de vídeo y audio MPEG-1 de Hitachi, y estarán dirigidos a las aplicaciones "Set Top Box", de disco vídeo digital y tarjetas gráficas vídeo PC. En un principio, Hitachi ofrecerá un decodificador audio y vídeo MPEG-2 de un sólo chip que fue definido en cooperación con varios fabricantes "Set Top Box" en instalaciones beta. Los futuros productos MPEG-2 estarán basados en estos primeros productos MPEG-2 y el motor RISC SuperH de 32 bits de Hitachi.

Para más información:
Hitachi Europe
Pedro Aparicio
Tel: (91) 767 27 82 y (91) 767 27 92

NUEVA VERSIÓN DE WORDPERFECT PARA MACINTOSH

Novell anuncia la disponibilidad de la versión 3.5 del procesador de textos WordPerfect para Macintosh. Esta versión ofrece nuevas características a los usuarios de ordenadores Macintosh, entre las que destacan Make It Fit, marcadores con "hot links", la posibilidad de escuchar los textos escritos mediante la tecnología MacinTalk de Apple y capacidades relacionadas con Internet.

WordPerfect 3.5 para Macintosh incluye utilidades para la creación y mantenimiento de páginas Web. Los usuarios pueden convertir cualquier documento WordPerfect al formato HTML o ver cualquier documento de Internet (con el software de conexión a Internet apropiado) mediante el Netscape Navigator incluido. Además, cualquier combinación de gráficos, vídeo y fuentes pueden ser visualizados mediante Novell Envoy.

El programa está disponible en discos y CD-ROM al precio de 189 dólares. También existe la posibilidad de actualizarse desde versiones anteriores de WordPerfect, desde cualquier otro procesador de textos o desde Works o cualquier *suite* para Macintosh por 89 dólares.

Para más información:
Novell Spain
Daniel Toledano
Tel: (91) 577 49 41

MICROSOFT Y BAY NETWORKS ANUNCIAN UNA RELACIÓN DE DESARROLLO Y MARKETING CONJUNTOS

Microsoft Corporation y Bay Networks han realizado una alianza de desarrollo, soporte y *marketing* conjuntos que incorporará los "Routing Services" (BayRS) de Bay Networks en Windows NT Server de Microsoft. Ambas compañías están desarrollando juntas un Interfaz de Programa de Aplicación (API) de *routing* y un modelo de "Routing Table Manager" en Windows NT.

Microsoft y Bay Networks colaborarán para desarrollar soluciones completas de administración de red, además de realizar pruebas de productos para asegurar la interoperabilidad entre los "Routing Services" de Bay Networks y Windows NT Server, y entre las futuras versiones de los productos de ambas compañías.

El acuerdo incluye la formación cruzada y el cambio de equipo. Además, las dos compañías ofrecerán las ventas conjuntas y formación técnica a sus organizaciones de ventas y socios revendedores respectivos.

Para más información:
Bay Networks
Ricardo Miranda-Naón
Country Manager
Tel: (91) 742 50 13

LÍDERES DEL GIS FORMAN EQUIPO CON BENTLEY PARA OFRECER UNA SOLUCIÓN GIS ABIERTA

Cuarenta y cinco fabricantes independientes de software (ISDs) de Estados Unidos, Europa y Canadá han expresado su intención de proporcionar aplicaciones GIS para MicroStation GeoGraphics, un software completo de GIS basado en MicroStation de Bentley Systems.

La arquitectura programable del producto permite la creación de aplicaciones avanzadas mediante la utilización de MicroStation BASIC y el entorno de programación MDL. El producto proporciona más de 2500 funciones MDL, de las cuales 350 son específicas de GIS.

Para más información:
Bentley Systems
Jesús Pazos
Tel: (91) 372 89 75

INTEL MEJORA SUS BENEFICIOS GRACIAS AL PROCESADOR PENTIUM

La fuerte demanda de procesadores Pentium ha provocado un incremento del 41% en la cifra de negocio de Intel y un alza del 36% en las ganancias por acción para el segundo trimestre de 1.995, en comparación con el mismo periodo del año anterior. Respecto al primer semestre del presente año, la cifra de negocio aumentó en un 37% y las ganancias por acción se incrementaron en un 41% en comparación con el primer semestre de 1.994.

La cifra de negocio del segundo trimestre de este año alcanzó el récord de 3.890 millones de dólares, arrojando un beneficio neto de 879 millones de dólares. Las ganancias por acción de 0.99 dólares aumentaron con respecto a los 0.73 dólares del segundo trimestre del pasado año.

Intel ha incrementado su inversión de capital de 1.995 estimada en 300 millones de dólares alcanzando un total de 3.500 millones de dólares. El incremento se ha producido como resultado de una rápida conversión a nuevas tecnologías de proceso para microprocesadores de más velocidad y la adición de una capacidad superior para satisfacer la demanda de los clientes.

Para más información:
Intel Corporation Iberia
Paseo de la Castellana, 39
28046 Madrid
Tel: (91) 308 25 52
Fax: (91) 310 54 60

NOVELL LANZA UN CLIENTE NETWARE PARA WINDOWS NT

Novell ha anunciado la disponibilidad del cliente Netware para Windows NT. A partir de ahora los usuarios de Microsoft Windows NT pueden conseguir una total integración con Netware, incluyendo los servicios de directorio (NDS) de Netware 4.

El acceso integrado a NDS proporciona a los usuarios de Windows NT nuevas ventajas como un sólo punto de administración, y la navegación y gestión del entorno de directorios de Netware a través del nuevo interfaz gráfico de usuario de 32 bits de la utilidad de gestión del administrador Netware (NWAdmin).

Otras características incluyen: diferentes protocolos de red (IPX/SPX o TCP/IP) y acceso compartido a recursos y archivos con MS-DOS, Windows 3.x, OS/2, Macintosh y UNIX

Para más información:
Abanico/Hot Line
Liliane Chinyavong
Tel: (91) 594 43 53

BAY NETWORKS REVELA UN NUEVO PROGRAMA DE ALIANZA CON REVENDEDORES

Bay Networks acaba de anunciar el programa "Bay Networks Partner Alliance Program" (Programa de Alianzas con Socios de Bay Networks), un nuevo programa dirigido a incrementar las ventas del Canal Mundial en todo el mundo. El programa proporciona amplios recursos de soporte de formación, *marketing*, técnico y de ventas a los revendedores de Bay Networks.

El nuevo programa dispone de dos designaciones de socios: Bay Networks Authorized Partners (Socios Autoriza- dos de Bay Networks) y Bay Networks Enterprise Solutions Partners (Socios de soluciones de empresa de Bay Networks). Los primeros tienen experiencia en venta y soporte de soluciones de conectividad y/o routing para entornos pequeños y medianos. Los segundos tienen amplios conocimientos de red, sistemas o integración de aplicaciones, y tienen experiencia en venta y soporte de soluciones de red a nivel de empresa.

El Programa de Alianza con Socios ya está disponible para los socios revendedores en América del Norte, Europa, América Latina y Asia Pacífico.

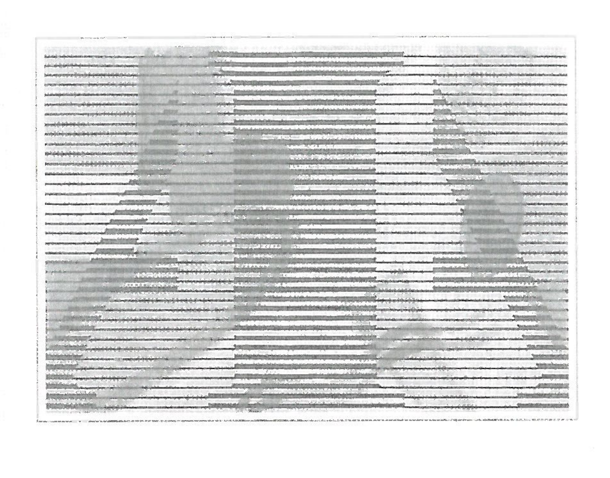
Para más información:
Bay Networks
Ricardo miranda-Naón
Country Manager
Tel: (91) 742 50 13

LIBROS

PRINCIPLES OF EXPERT SYSTEMS

Se trata de una obra de introducción a los sistemas expertos, creada para su utilización como libro de texto por estudiantes de informática. Proporciona una visión global de los sistemas expertos, exponiendo claramente los conceptos básicos y haciendo especial hincapié en la inferencia y en las técnicas de representación del conocimiento. El libro contiene multitud de ejercicios y ejemplos con su correspondiente codificación, centrándose en un problema de medicina cardiovascular. Los apéndices del libro están dedicados a los lenguajes Lisp y Prolog, orientados a los sistemas expertos.

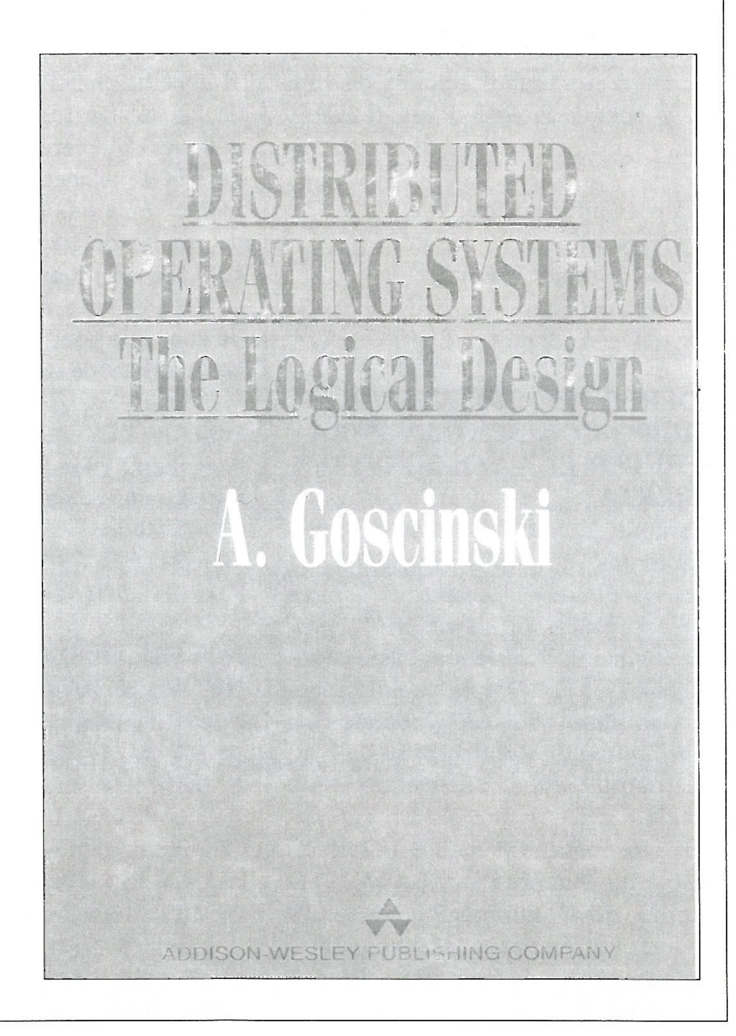
Editorial: Addison-Wesley
Autores: Peter Lucas y Linda Van Der Gaag
536 páginas
Idioma: Inglés
Precio: 5.200 ptas.



DISTRIBUTED OPERATING SYSTEMS. THE LOGICAL DESIGN

A diferencia de los sistemas operativos centralizados como UNIX, los sistemas operativos distribuidos han sido muy poco estudiados. El objetivo principal de esta obra es la de presentar las distintas áreas fundamentales para el desarrollo de un sistema operativo distribuido, junto con métodos y algoritmos específicos. Se exponen diversos temas como la sincronización, asignación de recursos, detección de interbloqueo, mensajes, sistema de ficheros y seguridad. Es un libro dirigido a lectores con conocimientos avanzados en el campo de los sistemas operativos, repleto de gráficos y referencias bibliográficas.

Editorial: Addison-Wesley
Autor: A. Goscinski
943 páginas
Idioma: Inglés
Precio: 6.500 ptas.





PARA AYER

En una tertulia radiofónica que escucho a menudo, hay un personaje que disfruta tomando el pelo a sus contertulios con sus sofismas y salidas de tono. Su gran habilidad reside en que, cuando el tema de debate le aburre o desagrada, se dedica a demostrar con total seriedad que aquello de lo que se está intentando hablar, sencillamente, no existe. Así ha ido dejando claro que la discriminación de la mujer no existe, que el terrorismo no existe o que los incendios forestales no existen. A veces me gustaría tomar prestadas sus habilidades para convencer a mi jefe de que los fallos de mis programas no existen, aunque los teléfonos de atención al cliente se saturen por ello. No me atrevo a intentarlo porque igual acaban diciéndome en la oficina de mi banco que mi nómina ya no existe.

En una ocasión en que el asunto de debate en aquella tertulia era especialmente oportunista, este tipo recurrió a una salida de gran mérito. Cuando se le preguntó su opinión sobre el tema, argumentó: "¿Ustedes piensan a menudo en la muerte? Quiero decir que si dedican ustedes al menos un minuto al día a reflexionar sobre la idea de que se van a morir. Es un pensamiento muy terapéutico que ayuda a poner las cosas en su sitio, y yo se lo recomiendo".

Lo que siguió fueron las señales horarias y un inserto de noticias, con lo que el resto de los participantes se libró de la responsabilidad de urdir una réplica suficientemente surrealista, pero aquellas palabras quedaron suspendidas sobre las ondas como un nubarrón de tormenta.

Mi jefe parece que no piensa a menudo en la muerte. O al menos, no en horas de trabajo. Al contrario, parece que quisiera zambullirse en ella por anticipado acelerando su corazón y los de quienes le rodeamos. Padece de uno de los males más comunes de nuestro oficio: la prisa. La inútil y condenada prisa.

Si realmente ésta es una revista para desarrolladores, no habrá lector que no sepa por propia experiencia que todas las cosas llevan siempre *más* tiempo del que uno predice. Y si uno predice *más*, llevan *más* todavía.

Todo informático sabe que esto es así. Por lo tanto, la estrategia consiste en presupuestar *menos* tiempo del razonable, de modo que, siendo el incremento constante, el tiempo final sea, precisamente, el razonable. Esta estrategia es de notable aplicación cuando se presupuesta el tiempo de *otros*.

El fenómeno se suele manifestar en cascada. Así, los gerifaltes de la empresa prevén un tiempo poco razonable para el proyecto. El jefe del departamento de informática gasta más tiempo del previsto en evaluar la viabilidad del proyecto, por lo que se ve obligado a presupuestar menos tiempo que menos tiempo del razonable para que el jefe de proyecto lleve el barco a buen puerto. El analista funcional gasta más tiempo del previsto en dibujar sus diagramas, dejando así menos tiempo que menos tiempo que menos tiempo del razonable para que el analista orgánico los mastique. El analista orgánico gasta más tiempo del previsto en estructurar un poquito todo aquello...

Aplíquese el algoritmo hasta alcanzar el último eslabón. Finalmente, el programador dispone de menos tiempo que menos tiempo que menos tiempo que menos tiempo del razonable para convertir en código el trabajo de sus congéneres. En la mayoría de los casos, para cuando el pobre currito recibe su orden, la fecha límite del primer tiempo poco razonable ya es una hoja de almanaque arrugada en la papelera. Y por este sencillo procedimiento, hemos descubierto el verdadero sentido de la frase favorita del jefe de cualquiera en este oficio: Esto es *para ayer*.

Ni que decir tiene que los procesos anteriores, a pesar de superar el tiempo asignado, están hechos deprisa y mal. Por consiguiente, el programador se enfrenta a una tarea mal evaluada, mal planificada y mal estructurada... que además debe estar terminada para ayer. Y para colmo, el pobre programata, último moco del pañuelo, no tiene a nadie debajo para asignarle menos tiempo del razonable.

Solución: volvamos al principio de las cosas. Pensemos de vez en cuando en la muerte, e intentemos que nuestros jefes también se hagan conscientes de su naturaleza efímera y perecedera. A lo mejor así recuperamos la cordura, dejamos de segregar adrenalina con plazos y prisas que no tienen ninguna trascendencia en el orden cósmico de las cosas, y hacemos lo que debimos haber hecho desde el principio: irnos todos a vivir al campo a comer nueces con pasas y a hablar con Dios.

"No sé lo que quiero, pero lo quiero YA"
Popular

VELOCIS 1.2.1

Fernando de la Villa

Bajo el nombre de Velocis se esconde un potente sistema gestor de bases de datos cliente/servidor orientado al desarrollo de aplicaciones con los lenguajes C y C++. Soporta SQL y brinda la posibilidad de combinar distintos modelos de bases de datos, lo que le convierte en una interesante herramienta de desarrollo.

Es posible que el nombre Velocis pueda confundir a algunos desarrolladores. En realidad Velocis es el nombre que recibe el producto Raima Database Server de Raima Corporation desde su versión 1.2.0. Se trata de un sistema gestor de bases de datos diseñado para proporcionar capacidades de manejo de bases de datos a las aplicaciones escritas en C y en C++. Los programas creados con Velocis son transportables de forma sencilla a varios sistemas operativos cliente diferentes, entre los que se encuentran MS-DOS, OS/2 y Windows. Una herramienta de desarrollo como Velocis posee multitud de características, entre las cuales se destacan las siguientes a modo de introducción:

- Soporte de SQL: lenguaje de manejo de datos (DML) y lenguaje de definición de datos (DDL) estático conforme con el estándar ANSI II, proporcionando acceso a datos puramente relacional en bases de datos con modelos combinados.
- Acceso múltiple a bases de datos: una aplicación puede abrir y utilizar más de una base de datos al mismo tiempo.
- Posibilidad de extender la función del servidor mediante los denominados módulos de extensión.
- Seguridad y administración del sistema centralizados.
- Librería de más de 100 funciones para obtener un control total de la base de datos.
- Utilidades interactivas y por lotes (*batch*) de acceso a la base de datos para una fácil manipulación de los datos.
- Proceso de transacciones.
- Recuperación automática de la base de datos en caso de fallo en el servidor.
- Soporte multiusuario para redes de área local (LAN) y ordenadores multiusuario.

Los programas creados con Velocis son transportables de forma sencilla a diversos sistemas operativos

LA FILOSOFÍA CLIENTE/SERVIDOR

Velocis es un producto pensado para trabajar en un entorno de red con el modelo cliente/servidor. Esta forma de trabajo implica la existencia de un proceso servidor y de uno o más procesos clientes. El proceso servidor se encarga de todos los accesos a la base de datos, validación de usuarios, gestión de errores y planificación. Los procesos clientes realizan sus operaciones y cuando necesitan acceder a la base de datos se limitan a lanzar una petición al servidor, que será atendida de la forma más adecuada. De esta forma se centralizan los accesos a la base de datos, minimizan-

do los errores y aumentando la seguridad del sistema.

COMBINACIÓN DE MODELOS

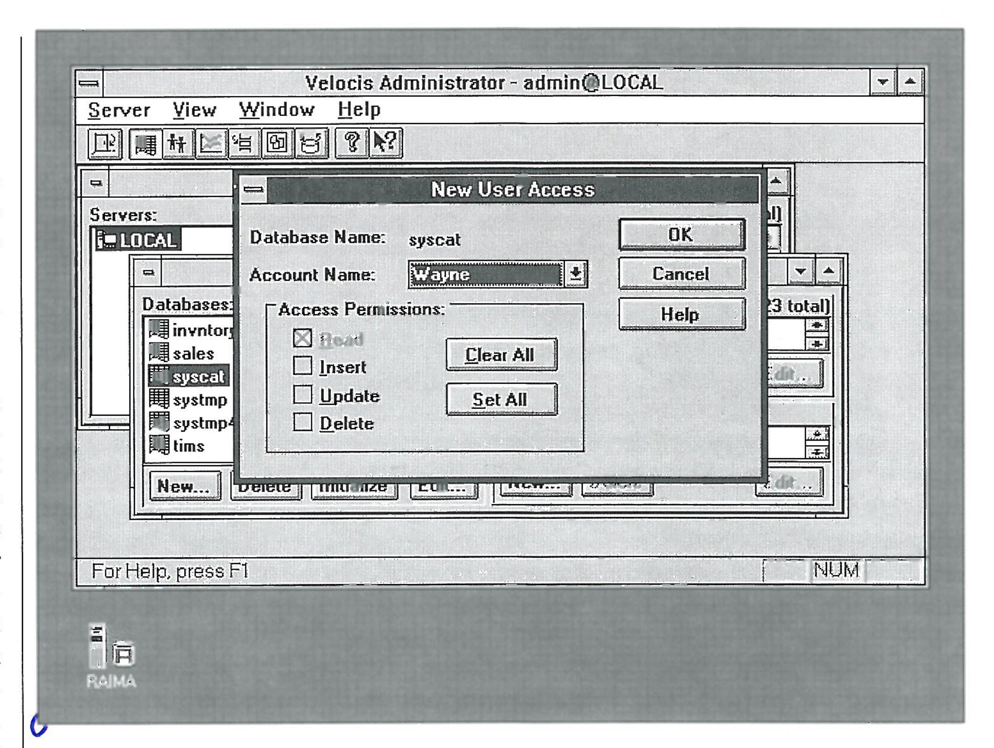
Una de las principales características que posee Velocis es la posibilidad de combinar el modelo de bases de datos en red y el modelo relacional. El modelo jerárquico, al ser un subconjunto del modelo en red, también es soportado.

La ventaja de combinar ambos modelos en una misma base de datos es la de quedarse con lo mejor de dos mundos en el modo en que más interese en el desarrollo de cada aplicación. Por un lado, las bases de datos relacionales tienen una estructura sencilla y por tanto son fáciles de utilizar. Sin embargo, se tiende a almacenar información de forma redundante lo que conlleva un desperdicio de soporte de almacenamiento y un mayor tiempo de acceso. Por otro lado, el modelo en red viene a ser lo contrario. Tiene un mayor rendimiento, precisa menos capacidad de almacenamiento y asegura una mayor integridad entre los datos. El precio que hay que pagar por estas ventajas sobre el modelo relacional es una estructura interna más compleja. La posibilidad de utilizar ambos modelos permite, por tanto, un diseño óptimo de la base de datos.

MÉTODOS DE ACCESO A LOS DATOS

Este sistema gestor de bases de datos ofrece tres métodos diferentes de acceso a la información: mediante índices, mediante referencias y de forma secuencial. Los tres pueden ser utilizados a la vez sin que el uso de alguno de ellos afecte a los otros dos. El único punto en común entre los métodos de acceso es el registro actual, que a su vez es el objeto por defecto para la mayor parte de las funciones de Velocis.

El primer método permite referenciar un registro mediante el uso de una clave. Los índices mantienen ordenados los registros en base a los valores de la clave, por lo que si no existe la clave especificada el registro actual apuntará al primer registro cuyo valor en la clave sea mayor que el especificado. Este método se corresponde con el modelo relacional de bases de datos.



El acceso mediante referencias permite moverse en varias direcciones a través de las conexiones establecidas entre los distintos tipos de registros de la base de datos. Está asociado con el modelo en red de bases de datos.

El último método permite recorrer la base de datos en el orden físico en el que se encuentran los registros de un determinado tipo, hacia delante o hacia atrás. Es lo más indicado si lo que se quiere es acceder a todos los registros de un mismo tipo sin importar el orden en el que se encuentran en la base de datos. Sin embargo, este método no permite realizar la inserción de un registro en cualquier lugar, debido a que utiliza posiciones físicas y esto daría lugar a serios problemas en la base de datos.

FUNCIONAMIENTO

Velocis está formado por varios componentes del sistema y utilidades cuyo funcionamiento será necesario comprender para poder desarrollar correctamente una aplicación. En la figura 1 se describen cómo interactúan los distintos elementos que componen Velocis

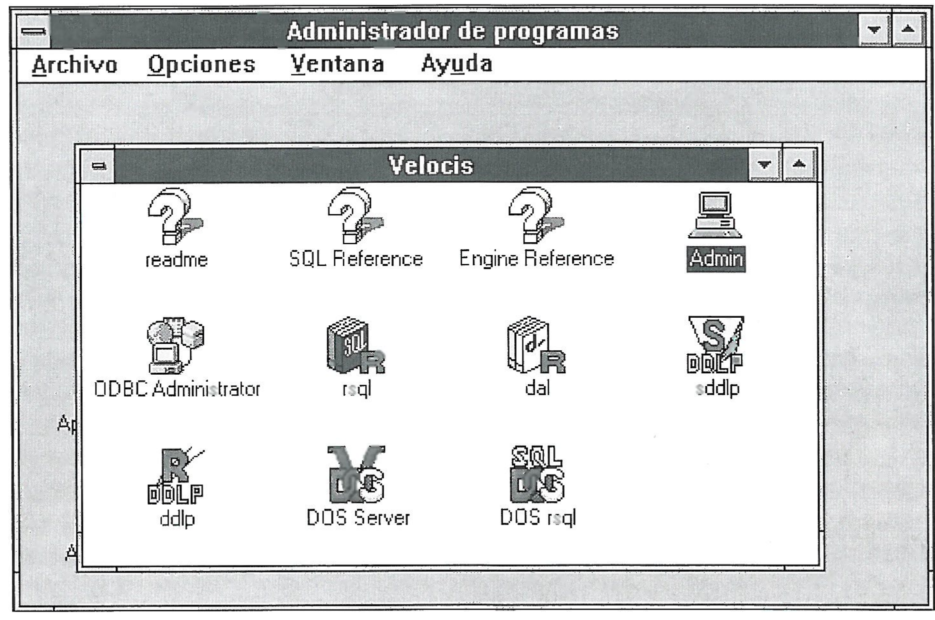
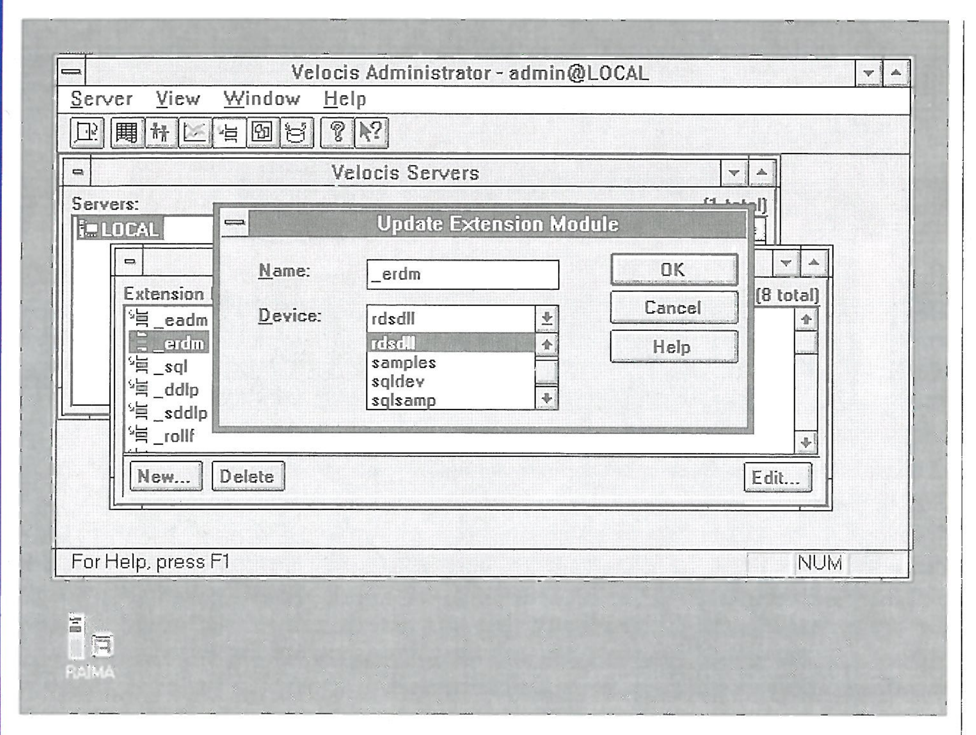


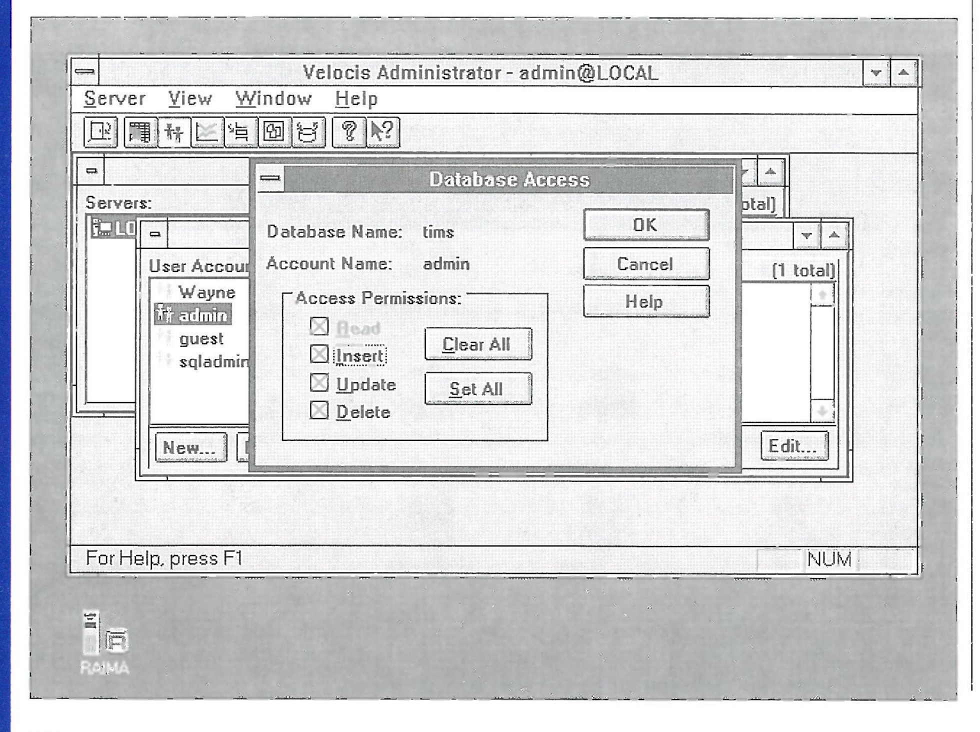
Figura 1: "Elementos que componen el sistema Velocis".



y a continuación se describen brevemente los mismos:

Servidor Raima de bases de datos: El servidor contiene todas las funciones de creación, acceso y definición de datos y bases de datos. Ofrece funciones que permiten la conexión y desconexión del servidor, mantenimiento de usuarios, apertura y cierre de bases de datos, bloqueos, transacciones, y funciones de seguridad y administración. Las aplicaciones se comunican con el servidor a través de la librería cliente y el procesador de comunicaciones de red (NCP) cliente. El servidor ofrece seis tipos de funciones a las aplicaciones:

- Funciones SQL SAG/ODBC CLI: El lenguaje SQL se utiliza a través de llamadas a funciones conformes con las especificaciones del SQL Access Group (SAG) y el interfaz del nivel de llamada (CLI) de Microsoft Open Database Connectivity (ODBC). El SQL soportado sigue la normativa X3.135-1989 ANSI SQL con mejoras de integridad.



- Funciones básicas del motor de bases de datos: Este API propietario proporciona las funciones necesarias para manipular bases de datos. Las funciones de este grupo se distinguen por el uso del prefijo d_ en su nombre.
- Funciones de administración: Realizan todas las tareas de tipo administrativo del servidor. Tienen el prefijo s_ y normalmente no hay necesidad de utilizarlas directamente. Para llevar a cabo estas operaciones se puede utilizar la utilidad admin, que será tratada más adelante.
- Funciones de acceso al diccionario: Son las funciones cuyo nombre comienza por c_. Sirven para obtener información del diccionario de la base de datos.
- Funciones de módulo de extensión: Estas funciones permiten realizar llamadas a funciones de módulo de extensión que hayan sido añadidas al servidor.
- Funciones del gestor de objetos Raima: El gestor de objetos Raima es una librería de clases que añade funcionalidad de orientación a objetos a una base de datos. Está escrito enteramente en C++ y utiliza Velocis como motor de acceso a datos.

Admin: Se trata de una utilidad manejada mediante un interfaz gráfico de usuario que permite la realización de la mayoría de las tareas administrativas en el servidor.

Dal: Su nombre proviene de Database Access Languaje (lenguaje de acceso a base de datos). Es un lenguaje interpretado que incluye todas las funciones básicas del motor de base de datos como comandos. Incorpora variables y construcción de bucles, por lo que es ideal para la realización de pruebas de funciones previas a la codificación en la aplicación.

Ddlproc: Esta utilidad es el procesador del lenguaje de definición de base de datos. Su misión es compilar el diseño DDL de la base de datos, generando un diccionario de base de datos que pueda ser procesado por el motor de Velocis. Dicho diccionario contendrá las declaraciones de los elementos de la base de datos y la organización de la misma. También produce un archivo cabecera de lenguaje C para que las

aplicaciones puedan acceder a los datos con mayor facilidad mediante las estructuras y constantes creadas automáticamente.

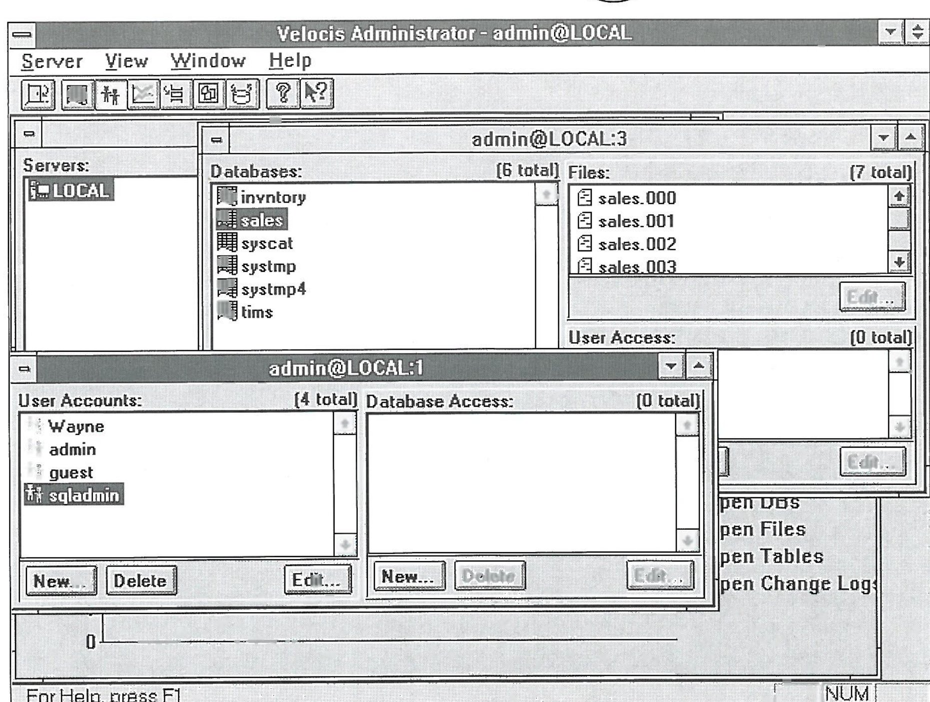
Rdsadm: Es el equivalente en modo texto de la utilidad admin.

LA UTILIDAD ADMIN

Se trata de una aplicación Windows que merece una atención especial por ser la encargada de facilitar las tareas de administración del sistema. Para poder operar con ella es preciso conectarse al servidor mediante la opción Login del menú Server. Se conecta con el nombre del usuario y su palabra clave. En Velocis existen dos tipos de usuarios: administradores y convencionales. Los administradores poseen todos los privilegios de acceso, mientras que los usuarios convencionales tienen limitadas ciertas operaciones. Según el tipo de usuario se podrá disponer o no de todos los comandos de admin. Después de conectar con éxito al servidor, aparecerá en pantalla una ventana con los servidores disponibles. Otras opciones del menú Server permiten cambiar los parámetros generales de configuración del sistema y del caché de disco para mejorar la eficiencia del servidor.

El menú View concentra casi toda la actividad de admin. Los principales comandos son:

- Accounts: Permite la creación, el borrado y la modificación de los datos referentes a los usuarios del sistema
- Databases: Operaciones de mantenimiento de bases de datos
- Devices: Especificación de dispositivos. En Velocis los dispositivos no son más que nombres que representan la localización física en la que se encuentran los archivos del sistema
- Extensions: Permite crear y modificar módulos de extensión del sistema
- Families: Sirve para asociar distintas bases de datos en familias. La utilidad principal de este mecanismo es la de realización de copias de seguridad y funciones de recuperación de errores
- Statistics: Muestra gráficos de tipo comparativo sobre la actividad de la base de datos

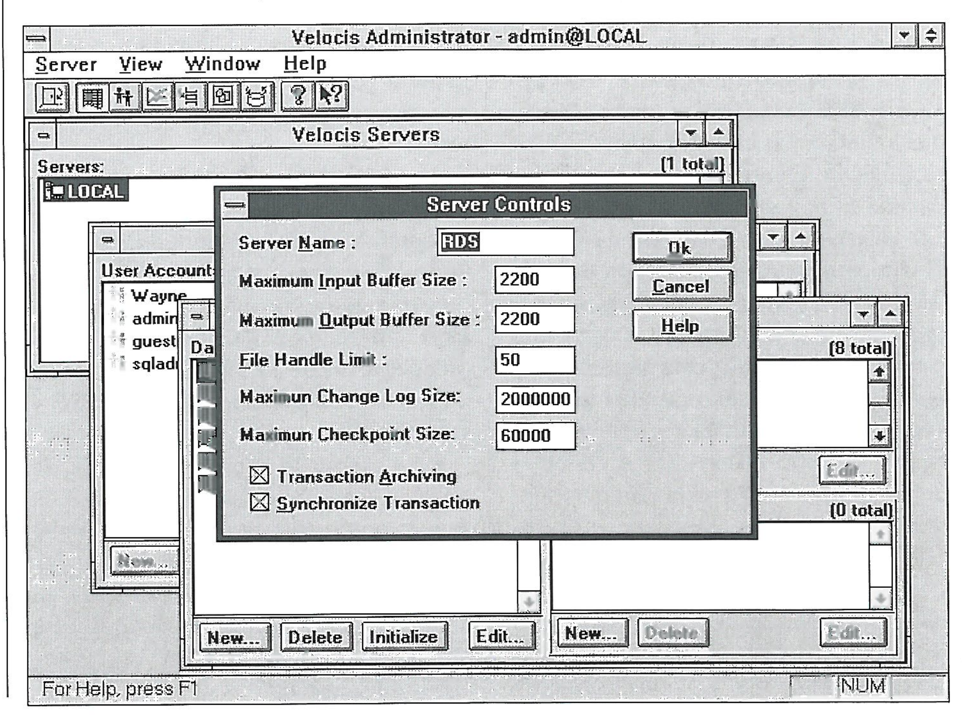


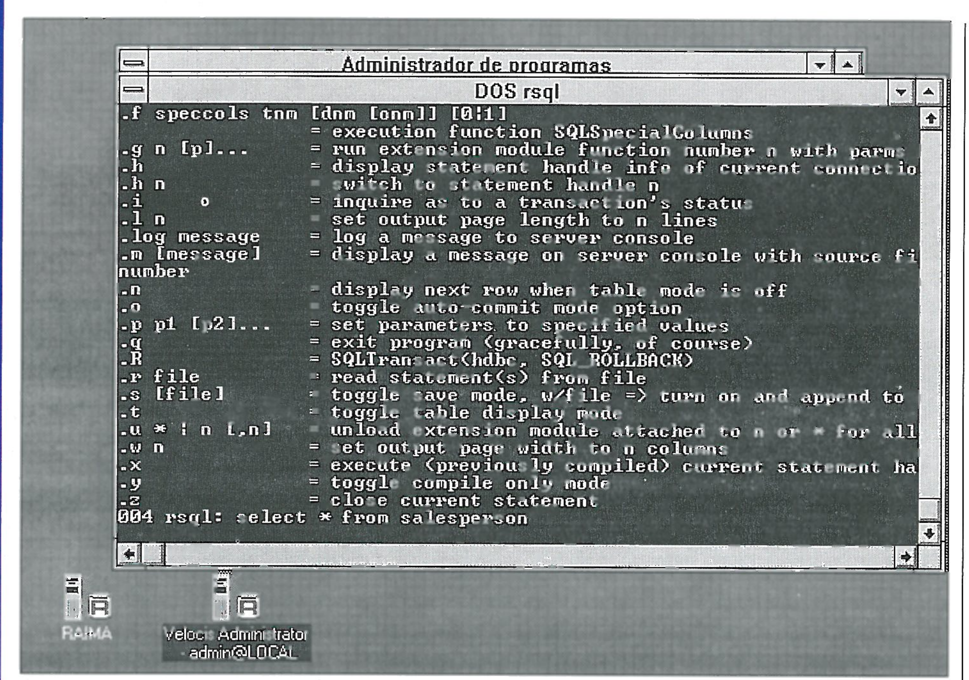
Cada una de estas opciones cambia la ventana actual, pero es posible la realización de comparativas abriendo varias ventanas a la vez con la opción New Window del menú Window.

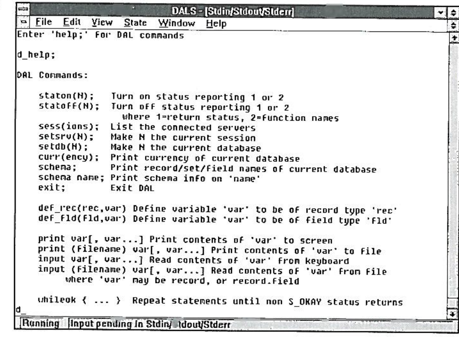
CREACIÓN DE UNA APLICACIÓN

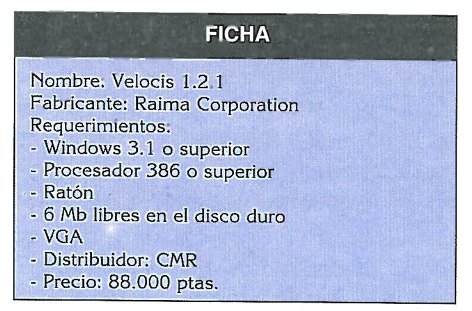
Una vez que se ha asimilado la forma en que interactúan los diferentes elementos que componen Velocis, se puede comenzar el desarrollo de una aplicación. Para simplificar se describirá a continuación el proceso básico de creación de una aplicación que no utiliza las funciones SQL del servidor.

En primer lugar se debe realizar el diseño de la base de datos, que debe quedar reflejado en un archivo de texto mediante DDL. A continuación se utiliza ddlproc para compilar el diseño. Si todo ha ido bien, se debe dar de alta en el servidor a la base de datos recién creada mediante las utilidades admin o rdsadm. Para cerciorarse de que todo es correcto hasta el momento se puede utilizar la utilidad dal para probar la








FICHA

Nombre: Velocis 1.2.1
Fabricante: Raima Corporation
Requerimientos:
- Windows 3.1 o superior
- Procesador 386 o superior
- Ratón
- 6 Mb libres en el disco duro
- VGA
- Distribuidor: CMR
- Precio: 88.000 ptas.

base de datos. Posteriormente se escribirá la aplicación en lenguaje C o C++, incluyendo los archivos cabecera rds.h y el generado por la utilidad ddlproc. Por último se compila la aplicación y se enlaza con la librería cliente de Velocis, con lo que se obtiene el ejecutable de la aplicación. En la figura 2 se describen de forma gráfica los pasos descritos.

UTILIZACIÓN DE SQL

El acceso al lenguaje de interrogación stándar no se realiza mediante el uso del propio lenguaje dentro del código de la aplicación. Como ya se ha dicho, Velocis está orientado al desarrollo de aplicaciones en lenguaje C y C++, por lo que SQL se utiliza mediante llamadas a funciones. Las sentencias SQL se especifican como parámetros de tipo cadena a las funciones de librería, son compiladas y ejecutadas de forma dinámica y los resultados producidos por cada sentencia pueden ser recuperados a través de llamadas a otras funciones de la librería. Una sentencia ya compilada puede ser utilizada repetidas veces sin necesidad de recompilarla de nuevo. Soporta referencias a variables del programa dentro de las propias sentencias, de manera que basta con cambiar el valor de determinadas variables y ejecutar la sentencia compilada para obtener nuevos datos. Los resultados obtenidos con una sentencia *select* son accedidos fila a fila, y es posible definir cursores para la inserción y borrado en posiciones concretas. También se proporcionan funciones para obtener información sobre la naturaleza de una sentencia compilada, haciendo posible saber el número de filas que componen el resultado y el nombre, tipo y longitud de cada una de las columnas resultantes. Como mejora al sistema se incluyen algunas funciones no contempladas en el estándar ODBC CLI, como una función que permite saber el tipo al que pertenece una sentencia compilada. Existe, sin embargo, una característica más potente que es la posibilidad de utilizar funciones definidas por el usuario (UDF) mediante las capacidades de programación basada en el servidor.

Las funciones SQL de Velocis tienen el prefijo SQL y se dividen en distintas categorías, entre las que figuran: conexión y desconexión a servidores, establecimiento de opciones SQL, compilación y ejecución de sentencias SQL, recuperación de la información obtenida por una sentencia, soporte ODBC y soporte UDF.

Para utilizar la funcionalidad SQL de Velocis desde una aplicación hay que seguir una serie de pasos. Se debe tener en cuenta que la mayoría de las funciones tienen que ser utilizadas en un orden predeterminado, aunque este orden sea bastante intuitivo. Los pasos son los siguientes:

1. Reservar espacio para el manejador del entorno con la función *SQLAllocEnv*.
2. Reservar espacio para el manejador de la conexión mediante la función *SQLAllocConnect*.
3. Establecer la conexión con el servidor de bases de datos. Para ello se utiliza la función *SQLConnect*.
4. Utilización de las sentencias SQL necesarias para la aplicación
5. Desconectar del servidor llamando a *SQLDisconnect*.
6. Liberar el espacio utilizado por los manejadores mediante las funciones *SQLFreeConnect* y *SQLFreeEnv*.

EN DEFINITIVA

Como se puede apreciar Velocis es un sistema gestor de bases de datos potente y bastante completo. Ofrece un alto rendimiento gracias a la inclusión de características como las extensiones de aceleración relacional (RACE) y el caché de disco. Es un sistema muy flexible debido a que ofrece la posibilidad de ampliar la funcionalidad del servidor, para ajustarlo a las necesidades concretas del desarrollador de aplicaciones. En cuanto a la documentación que acompaña al sistema, decir que es adecuada y está bien estructurada. Resumiendo: Velocis se perfila como una buena solución para muchos desarrolladores de aplicaciones de bases de datos.

OPTIMIZACIÓN DE CÓDIGO ENSAMBLADOR

Enrique de Alarcón

Los ordenadores aumentan geométricamente su potencia de proceso y los lenguajes de alto nivel son cada vez más potentes y rápidos.

Pese a eso todos los lenguajes tienen un punto en común, son superconjuntos de instrucciones en ensamblador (código máquina), y el programa compilado resultante, siempre poseerá mucho código que se podría eliminar y substituir por otro mucho más eficiente.

Todas las aplicaciones que necesitan velocidad de proceso (el ejemplo más claro son los videojuegos) tienen, por poco que sea, código escrito directamente en lenguaje ensamblador.

Pero no todo se soluciona por escribir en ensamblador. Unos fuentes ensamblador mal escritos, pueden dar como resultado código más lento que el generado por unos fuentes C compilados en un buen entorno de desarrollo como puede ser por ejemplo el moderno WATCOM C.

LAS INSTRUCCIONES ENSAMBLADOR

Los ciclos necesarios para la ejecución de una instrucción ensamblador dependen de 3 factores principales.

1-Los tiempos mínimos impuestos por la propia arquitectura del procesador:

Esto se comprende fácilmente: una multiplicación de 32 bits en un 386, necesita un mínimo de 9 a 38 ciclos de reloj para ejecutarse simplemente por la construcción física de su microcódigo (algoritmo interno de la CPU que da solución al problema). Esta regla sólo es válida en procesadores CISC, que se basan en instrucciones con microcódigo asociado. En procesadores RISC, cada instrucción posee una algoritmo de un sólo paso, y por lo tanto no tiene microcódigo asociado, todas se ejecutan en un sólo ciclo independientemente de su función.

2-De los tipos de parámetros que posea:

Prácticamente todas las instrucciones ensamblador pueden operar directamente tanto con variables de memoria como con registros de la propia CPU (zonas almacenamiento temporal de datos de la CPU). El uso de registros siempre resulta más rápido. Esto es debido a que la memoria de los registros está dentro de la propia CPU. Por ejemplo MOV AX,BX, pasa el contenido del registro BX a AX. Al ser los dos registros de la CPU, se accede a ellos muy rápidamente, y la instrucción se ejecuta por lo tanto en menos tiempo. Si hubiésemos puesto un operador de memoria como por ejemplo MOV AX,DS:SI, primero debemos extraer el valor de la posición de memoria apuntada por conjunto DS:SI (memoria RAM que normalmente es muy lenta de acceder), y tras obtener el valor, realizar la transferencia del valor al registro de destino.

3-De la longitud (en bits) de los operadores que usemos. Siempre tardará más una multiplicación de números de 32 bits que otra con números de 8 bits. Al igual que en la multiplicación, todas las instrucciones siguen esta regla.

FUNDAMENTOS DE LA OPTIMIZACIÓN

Las optimizaciones, se basan en el menor acceso posible a memoria, en el

00101010010
00101010010
10101001010
01010101001
01010101010
101000100111
10101010101
00101101010
110101101010

El ensamblador es el lenguaje usado por la CPU, y por lo tanto el más rápido, pero aun así, también se debe saber cómo optimizar al máximo los fuentes de éste lenguaje para sacar un buen rendimiento de la máquina.

uso de las instrucciones que necesitan menos ciclos de ejecución y en el uso de 'trucos' que hagan el código lo más corto posible y eficiente. A partir de estas tres premisas, podemos dar toda una serie de reglas en las que podemos basarnos para realizar nuestras optimizaciones en los fuentes.

A continuación listaremos todas las reglas...

REGLAS DE OPTIMIZACIÓN

-Substituir todas las instrucciones *MOV REG*,0 (donde *REG* es cualquier registro de *CPU*) por *XOR REG,REG* o *SUB REG,REG.* Por ejemplo, *MOV EAX,0* sería *XOR EAX,EAX* o *SUB EAX,EAX.* Fijarse que los dos parámetros son el mismo registro. De esta manera la instrucción requiere menos bytes para ser definida y elimina la necesidad de darle un valor inmediato al registro, por lo que se acorta la instrucción y se hace más rápida al usar sólo registros.

-Cuando debamos realizar una multiplicación o una división de un número por un múltiplo de 2, usar las instrucciones *SAL* y *SAR* respectivamente. Por ejemplo, multiplicar el registro *EAX* por 4, se puede hacer con la instrucción *SAL EAX*,2 que rota 2 bits hacia la izquierda el registro *EAX.* Esta instrucción requiere para ejecutarse 3 ciclos de reloj frente a los 9-38 ciclos que necesita la instrucción *IMUL*, ello sin contar que ahorramos líneas para poner el segundo operador en otro registro para multiplicar *EAX* por ejemplo *EBX.*

-En bucles que se ejecutan cientos de veces, no usar variables de memoria para ir almacenando valores temporales y otros.
Por ejemplo:

```
Variable SUM= valor constante
sub eax,eax
mov ecx,1000000
BUCLE1:
 add eax,SUM
loop BUCLE1
```

Aquí se ve que la suma *ADD EAX,SUM* se ejecuta 1000000 veces y cada vez requiere 6 ciclos, que da un total de 6000000 ciclos, en cambio si damos antes el valor de *SUM* a un registro (Ej *EBX*), la instrucción queda *ADD EAX,EBX* que sólo necesita 2 ciclos, o sea un total de 2000000 que representa un ahorro de 2/3 de ciclos.

-Aprovechar las informaciones que dejan algunas instrucciones en los flags. Pudiendo ahorrar líneas en saltos condicionales y otros. Por ejemplo, incrementar un registro y comprobar si queda con el valor 0:

```
inc eax
cmp eax,0
je ETIQUETA
```

Como la instrucción *INC EAX* modifica el flag *ZF* (Indicador 0), ya tenemos el flag condicional modificado y no necesitamos *CMP EAX,0.* El código quedaría como sigue:

```
inc eax
jz ETIQUETA
```

-Aprovechar las comparaciones para más de un salto. Por ejemplo, si comparamos ax con 1, podemos deducir si el número de ax tiene signo, si es negativo, igual mayor o menor.:

```
cmp ax,1
js NUM_NEGATIVO
jb NUM_MENOR
je NUM_IGUAL
ja NUM_MAYOR
```

-Modificar directamente el código mediante la escritura de bytes en puntos concretos para eliminar instrucciones condicionales del interior de bucles muy grandes (críticos). De esta manera podemos modificar el código en tiempo real.

Para ello, debemos conocer cuál es la codificación binaria de las instrucciones y considerar que no todas ocupan el mismo número de bytes. En caso de que queden 'huecos' tras las modificaciones donde estaba el anterior código, debemos rellenarlo con tantas instrucciones NOP como bytes hayan quedado libres. Además, en este caso, debemos ver si compensa el tiempo muerto de ejecución de múltiples NOP que nos queda, y que no hacen nada al haber dejado las condicionales sin modificar. El cálculo deberá hacerlo el programador y considerar si vale la pena o no.

-Crear siempre procedimientos que recojan los parámetros de entrada mediante los registros de la CPU y no mediante la pila. De esta manera, en funciones no muy complejas, y si adaptamos bien el código, podemos empezar el algoritmo con valores en registros. Basándonos en la pila tendríamos que realizar muchas instrucciones PUSH, y transferencias entre Pila(memoria), y registros antes de usarlos, lo cuál resulta mucho más lento.

-Cuando usemos registros/variables como acumuladores temporales, usar siempre los tamaños más pequeños necesarios. Por ejemplo si sabemos que un acumulador nunca poseerá valores mayores a 2^16, no se debe usar un registro de 32 bits como EAX, pues en caso de tener que operar con el registro obtendremos instrucciones más largas en bytes y en ciclos. En su lugar, usaremos un registro AX de 16 bits.

-Cuando tengamos que realizar saltos a procedimientos condicionales, según (por ejemplo) el valor obtenido a partir de un cálculo, no usaremos saltos a etiquetas que apunten a llamadas CALL, sino saltos directos a procedimientos almacenando previamente la dirección de retorno en la pila. Ejemplo:
Suponemos que siempre la resta dará *entre 0 y 3:*

```
sub ax,bx
cmp ax,1
jb SETQ_0
je SETQ_1
cmp ax,3
jb SETQ_2
je SETQ_3
SETQ_0:
  call PROCED0
  jmp FIN
SETQ_1:
  call PROCED1
  jmp FIN
SETQ_2:
  call PROCED2
  jmp FIN
SETQ_3
  call PROCED3
 FIN:
...Resto del código...
```

Este código es antiestético además de innecesario en cuanto a longitud. Aplicando nuestra solución el código quedaría:

Suponemos resultado siempre entre 0 y 3

```
sub ax,bx
push word ptr ETIQUETA_NEXT
cmp ax,1
jb PROCED0
je PROCED1
cmp ax,3
jb PROCED2
je PROCED3
ETIQUETA_NEXT:
...Resto del código...
```

-Usar trasferencias de bloques de memoria siempre en unidades tan grandes como podamos:

Por ejemplo, para limpiar una zona de un buffer de pantalla de 13111 bytes con un valor X, no se debe usar la instrucción stosb, sino dividir la longitud total entre 4 y limpiar todo lo que podamos en bloques de 4 bytes.

En el ejemplo, si el valor es de 8 bits, se debe repetir el mismo valor en los 4 componentes de 8 bits del registro *EAX*.

Supongamos que inicialmente *ECX*=Longitud bloque, que *EAX*= color de relleno y que *ES:EDI*=dirección destino (buffer) . El código sería:

```
mov dl,cl
shr ecx,2
and dl,11b
rep stosd
mov cl,dl
rep stosb
```

Aunque en el código quedan más líneas, tardaremos _ menos de tiempo en realizar la transferencia del valor a todo el buffer. Fijémonos que para obtener el número de bloques múltiplos de 4 bytes que hay, sólo tenemos que dividir *LONGITUD_BUFFER/4 = SHR ECX,2* y el resto de la división lo obtenemos cogiendo los 2 bit más bajos del número con la longitud completa.

-Mientras no debamos realizar operaciones extras para evitarlo, debe usarse siempre el menor número de registros para operaciones temporales. Cuanto menos alteremos registros, menos *PUSH y POP* necesitaremos en procedimientos que hagan llamadas.

Por ejemplo:

```
mov ax,MEM_SEG
mov es,ax
mov bx,LONG_C
mov VALOR2,bx
...código...
```

Lo podemos dejar en:

```
mov ax,MEM_SEG
mov es,ax
mov ax,LONG_C
mov VALOR2,ax
```

-Crear tablas con valores precalculados cuando sean complejos y todos los valores posibles no demasiados. Por ejemplo, podemos tener todos los senos y cosenos calculados en una tabla con incrementos de 0.1pi radianes entre valor y valor, con lo cuál, nos cabrán todas las combinaciones en unos 12 Kb., y nos ahorramos entre 100 y 700 ciclos de proceso del coprocesador en cálculos por cada seno o coseno que queramos calcular.

-Usar registros como emuladores de decimales, con lo cuál nos ahorramos el usar el coprocesador, y aumentamos el rendimiento de los algoritmos, por ejemplo, para hacer una suma incrementa de una coordenada X a través de una Y que se incrementa de 1 en 1, haríamos como sigue:

```
mov eax,Y
sal eax,16
cdq
mov ebx,X
idiv ebx
```

El número obtenido en *EAX* contiene la parte entera en los 16 bits superiores, y la parte decimal en los 16 inferiores(*AX*). De ésta forma tenemos una precisión equivalente a unos 5 dígitos decimales para la X.

ULTIMAS CONSIDERACIONES

Además de todos estos consejos, es necesario siempre tener al lado una tabla con los tiempos de ejecución de todas las instrucciones según el tipo de CPU en que nos basemos y de los parámetros posibles que acepte la instrucción.

Además, y también muy importante, a partir del 486 y sobre todo del Pentium, se debe saber programar teniendo en cuenta la capacidad de estos micros de la ejecución simultánea de más de una instrucción, lo cuál puede incrementar la velocidad de proceso en un porcentaje bastante elevado.

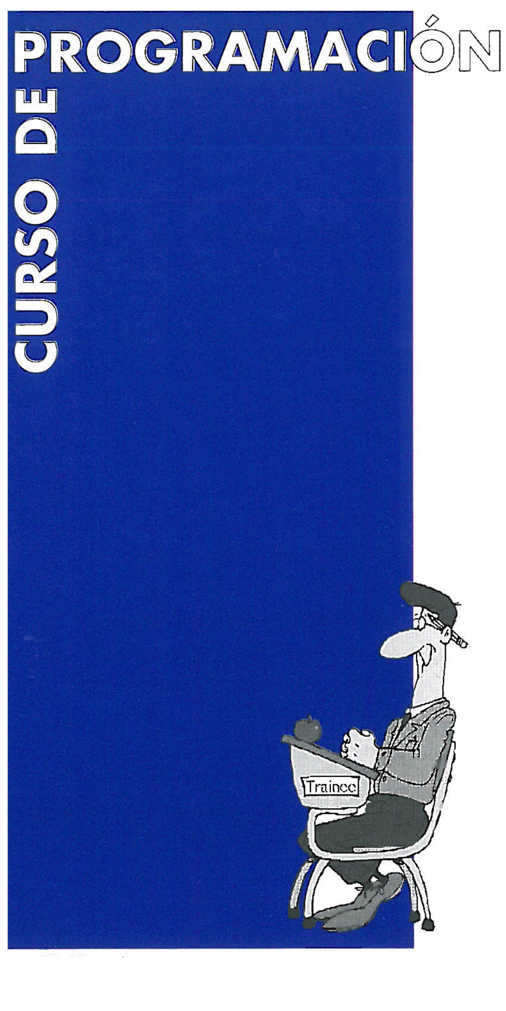
BIBLIOGRAFÍA

*Programación del 386/387. Manual de referencia y técnicas avanzadas de programación para diseñadores de sistemas, programadores y usuarios avanzados. John H. Crawford / Patrick P. Gelsinger.
Anaya Multimedia.
*8088-8086/8087 Programación ENSAMBLADOR en entorno MS-DOS, Miguel Angel Rodriguez Roselló. Anaya Multimedia.

RESUMEN

A continuación listamos un resumen de todas las reglas.

1-Usar restas o Xor para evitar asignaciones a 0.
2-Cambiar multiplicaciones múltiples de 2 por shr/sar, shl/sal...
3-Aprovechar la información que dejan las instrucciones en los flags para saltos condicionales sin uso de comparaciones.
4-Usar más de un salto por comparación.
5-Modificación de código en tiempo real para evitar instrucciones condicionales en bucles muy críticos.
6-Paso de parámetros en registros.
7-Uso de registros/variables lo más pequeños posibles
8-Saltos múltiples a procedimientos usando jmp y call.
9-Control de bloques de memoria en bloques de 4 bytes para aprovechar las capacidades 32 bits de los micros 386 y superiores.
10-Usar el menor número de registros necesarios.
11-Crear tablas con valores precalculados.
12-Usar registros de la CPU como emuladores de decimales.



ESTRUCTURA DE DATOS ARRAY (II)

José C. Remiro

Los arrays multidimensionales son menos utilizados que los arrays de una dimensión, pero resultan adecuados para implementar determinadas estructuras de datos. La representación de cadenas de caracteres es muy similar a los arrays, por lo que se desarrollarán en este artículo.

Los *arrays* hasta ahora utilizados presentaban una única dimensión, es decir, para acceder a un elemento del *array* bastaba con indicar el nombre del *array* junto a una posición, la que éste ocupaba dentro de la estructura, así un *array* puede verse como una lista de elementos ordenados.

Hay situaciones en que es útil disponer de *arrays* con varias dimensiones, entendiendo por este término la accesibilidad a elementos por medio de varios índices. Supongamos que se desea representar las posiciones que ocupan las fichas en el juego de las damas en un tablero de ajedrez. Una solución sería representar el tablero mediante un *array* unidimensional de 64 posiciones (recordemos que el tablero de ajedrez tiene 8 filas de 8 componentes). Esta representación, en principio válida, ocultaría la visión que poseemos en la vida real del tablero y dificultaría las operaciones que posteriormente se tendrían que realizar sobre él. Afortunadamente, los lenguajes de programación nos permiten disponer de *arrays* de varias dimensiones, por lo que podríamos representar el tablero de forma similar a como en realidad se dispone. El tablero consta de 8 filas, cada fila se compone de 8 elementos, se puede acceder a cada elemento del tablero indicando la fila y la columna donde se encuentra. Por tanto, ahora es necesario para acceder a cada elemento de la estructura dos índices, uno correspondiente a la fila y otro a la columna (cuadro 1).

El anterior ejemplo es un caso particular, de dos dimensiones, de un *array* multidimensional. Podríamos disponer de estructuras con mas dimensiones, aunque lo mas usual es utilizar *arrays* de hasta tres dimensiones.

Se aumentará el seudocódigo hasta ahora utilizado para poder declarar *arrays* multidimensionales. Para ello bastará con hacer seguir al nombre de la variable, la palabra reservada *array* y entre paréntesis, separados por comas el tamaño de los índices. Por ejemplo, para declarar el anterior tablero podríamos hacerlo de la siguiente forma:

tablero: array (8,8) carácter

Los componentes de estos *arrays* se pueden utilizar de igual forma que los *arrays* unidimensionales, pero para acceder a cada una de los componentes básicos necesitaremos tantos índices como los de la declaración. En nuestro caso particular necesitaremos dos índices, uno para indicar la fila a la que se accede y otro para acceder a la columna correspondiente, así, el elemento *tablero (5,2)*, se refiere al elemento situado en la fila 5 , columna 2. Este elemento será del tipo básico carácter, pudiendo aparecer en cualquier situación en que pueda hacerlo cualquier variable de este tipo. Debe tenerse cuidado en la especificación de índices, pues la alteración en el orden de los índices provocará el acceso a elementos diferentes del *array* (es diferente el elemento *tablero (5,2)* que el elemento *tablero (2,5)*).

Algunos lenguajes de programación permiten el acceso a las filas de un *array* multidimensional, como tales. Por ejemplo, supongamos que deseamos trabajar con la sexta fila de la variable *tablero*, si el lenguaje lo permitiera, se

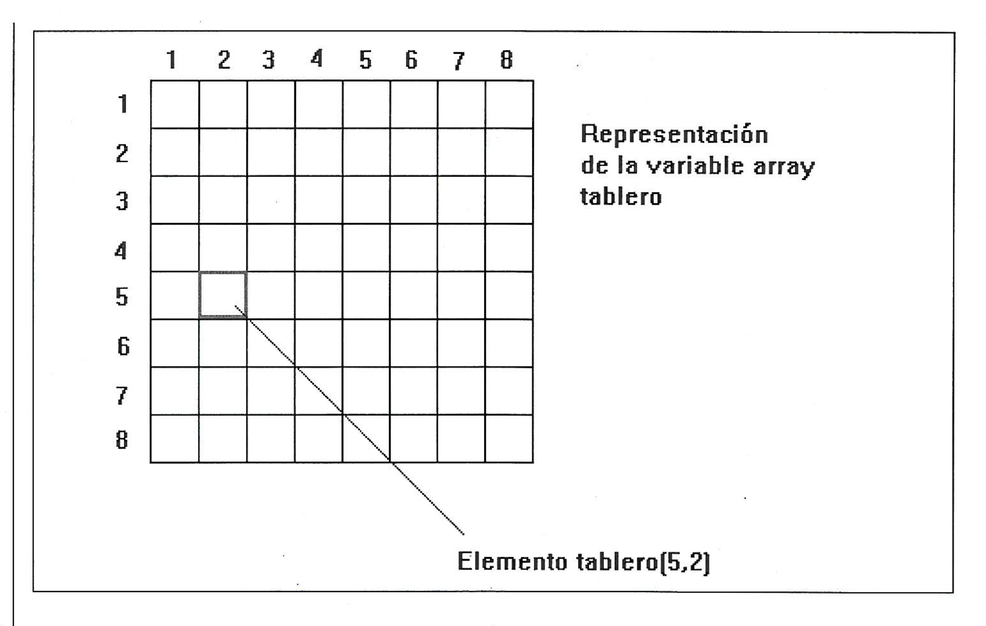
podría referenciar a dicha fila mediante *tablero(6)*, ahora bien, esta fila no sería un elemento básico del *array* multidimensional (tipo *carácter*), pues falta especificar un índice para obtener la columna del elemento, *tablero(6)* especificaría un *array* unidimensional de 8 componentes que coincidiría con los elementos *tablero(6,l)*, con *l*, variando entre 1 y 8. *Tablero(6)*, podría utilizarse en cualquier situación en la que estuviera permitido utilizar un *array* con 8 componentes de tipo carácter.

UN EJEMPLO

El cuadro 2 contiene el procedimiento *calcula_tpc*. Este procedimiento utiliza el parámetro *vend_prod*, un *array* bidimensional que representa las ventas de una serie de vendedores (se dispone de un total de 4 vendedores), para cada tipo de producto (se dispone de 20 productos), por tanto representa 80 informaciones básicas: por ejemplo, la componente *vend_prod(8,2)* contiene la cantidad que ha vendido el vendedor 2 del producto 8. A partir del parámetro *vend_prod*, se calcula en el parámetro *tanto_pci* el tanto por ciento que cada vendedor ha vendido de cada uno de los productos. Para ello se utiliza una variable auxiliar *tot_vend_prod*, donde para cada producto se almacena el total vendido para cada producto, de esta forma, la posición *tot_vend_prod (5)*, por ejemplo, contendrá la suma de la columna del parámetro *vend_prod* de las posiciones 5, es decir, la suma de los elementos *vend_prod(5,l)*, con *l* variando entre 1 y 4.

REPRESENTACIÓN EN MEMORIA DE LOS ARRAYS

Los lenguajes representan un nivel de abstracción superior al de la máquina, de tal forma que no es necesario comprender la implementación de un determinado lenguaje sobre la máquina, y no es necesario para construir, comprobar ni verificar el programa. De todas formas conocer como se implementan internamente la estructura *array* dentro de la máquina es una ayuda para que el programador tome decisiones adecuadas sobre el diseño del programa y de las estructuras de datos.



El *array* debe representarse en memoria de tal forma que el acceso a sus componentes sea lo más rápido posible. Si se supone que el tipo básico del *array* ocupa *tc* palabras (por palabra se entenderá la unidad mas pequeña de la memoria física que puede accederse de manera individual) y que la primera componente del *array* se encuentra situada en la dirección de memoria *mi*, la componente de posición *j*, se encontrará situada en la dirección *mi + tc x j*. Mientras *tc* sea un entero, no hay ningún tipo de problema, pues esto implica que la información a representar se ajusta perfectamente a las zonas de memoria, pero si esto no ocurriese, la representación en memoria del *array*, estaría desperdiciando espacio. El compilador normalmente ajusta *tc* al menor entero mayor que *tc*. Una solución al anterior desperdicio de memoria es utilizar una técnica de empaquetamiento, que reduzca el espacio utilizado a costa de complicar el cálculo de la posición de las componente. Existen lenguajes de programación que permiten definir de manera explícita el empaquetamiento de *arrays*, con el consiguiente ahorro de memoria.

LENGUAJES Y ARRAYS MULTIDIMENSIONALES

Se describirá brevemente como tratar los *arrays* multidimensionales en los lenguajes Pascal, QBasic y C.

En Pascal, los *arrays* multidimensionales se declaran de manera parecida a como se comentó en el anterior artículo, evidentemente con la incorporación de los índices necesarios, por ejemplo:

```
const
 N = 10;
 M = 20;
type
 tabla = array[1..N, 1..M] of integer;
 columna = array [1..M] of integer;
 fila = array [1..N] of columna;

var
 T1:tabla;
 T2: fila;
....
```

En este ejemplo, se han declarado dos variables que parecen tener diferentes tipos pero que básicamente representan un *array* de dos dimensiones, compuestos de 200 elementos de tipo integer. La variable *T1* se utilizará de la siguiente forma *T1[i,j]* para seleccionar sus elementos, mientras que la selección los elementos de la variable *T2*, es como sigue *T2[i][j]*. Ambos métodos para definir un *array* de más dimensiones son posibles y se puede seleccionar el más adecuado o el que resulte más claro. Se debe recordar también que para definir un *array* en Pascal no es necesario que se declare como tipo (aunque es una buena práctica), se puede realizar en la sección dedicada a las variables.

En Qbasic la declaración es aún mas sencilla, por ejemplo:

Dim tabla (10, 12) as integer

```
procedimiento calcula_tpc (vend_prod: array (20,4) entero;
                           var tanto_pci: array(20,4) real);

 i, j: entero;
 tot_vend_prod: array(1,20) entero;

inicio
 {proceso de inicialización del array de acumuladores}

 desde i = 1 hasta 20
   tot_vend_prod(i) = 0

 {cálculo de los totales por producto}

 desde i = 1 hasta 20
    desde j = 1 hasta 4
      tot_vend_prod(i) = tot_vend_prod(i)+ vend_prod(i,j)

 {cálculo del tanto por ciento para cada vendedor y producto}

 desde j = 1  hasta 20
    desde j = 1 hasta 4
 {comprobación de una posible división por cero}
       si tot_vend_prod(i) = 0
          entonces
            tanto_pci(i,j) = 0
          en otro caso
            tanto_pci(i,j) = 100 * vend_prod(i,j) / tot_vend_prod(i)
fin
```

Dim amd (2 to 7, 3 to 5, 4 to 6) as long

La variable *tabla* tiene elementos de tipo *integer*, dispone de 143 elementos en total (recordar que en este tipo de descripción de índice el primer valor es 0). El acceso a las componentes de los elemento es como sigue, *tabla(i,j)*, donde el valor de *i* debe estar comprendido entre 0 y 10, y el valor de *j* debe estar comprendido entre 0 y 12.

En el caso de la variable *array amd* se trata de un *array* tridimensional, se compone de 54 elementos de tipo long, para acceder a sus elementos se procede como sigue, *amd(i,j,k)* donde los valores *i*, *j* y *k* deben estar comprendidos entre los límites de declaración de la variable.

En el lenguaje C, la declaración de un *array* multidimensional se realiza precediendo a la variable *array* del tipo de datos elementales que componen el *array*, y a continuación y entre corchetes el número de elementos de la fila y de la columna, por ejemplo:

int tabla [3][5]

en el anterior ejemplo se ha definido una variable *array* que dispone de 3 filas y cada una de estas filas se compone de un *array* de 5 elementos. En C, un *array* multidimensional es un *array* unidimensional por definición, en el que cada uno de sus elementos es un *array*. Así, los accesos a las componentes se denotan de la siguiente forma *tabla[i][j]*, en lugar de la notación más general utilizada en otros lenguajes. En caso de que se desee transferir un *array* a una función, la declaración del parámetro de la función deberá incluir el número de columnas, pero no el número de filas, ya que lo que se transmite es realmente un apuntador (como se comentó en un artículo anterior).

CARACTERES

Hoy en día, todos los lenguajes de programación se adecuan para procesar datos alfanuméricos, puesto que la función de los ordenadores está dirigida principalmente a la gestión de información con formato texto (bases de datos, procesadores y editores de texto...).

Internamente, el ordenador almacena cualquier información mediante secuencias de ceros y unos, mientras que la información representativa básica para el usuario sigue siendo el conjunto de símbolos que utiliza normalmente, es decir, letras, números y símbolos especiales. A la correspondencia entre la representación interna que tiene el ordenador de estos símbolos y los símbolos normalmente utilizados (y alguno más) se le denomina código de caracteres. Estos códigos suelen estar estandarizados. Los dos códigos más importantes actualmente son EBCDIC y ASCII, siendo el código ASCII el utilizado en todos los ordenadores personales actuales.

CADENAS DE CARACTERES

Se distinguirá a continuación entre un literal alfanumérico y una variable capaz de almacenar tal tipo de información.

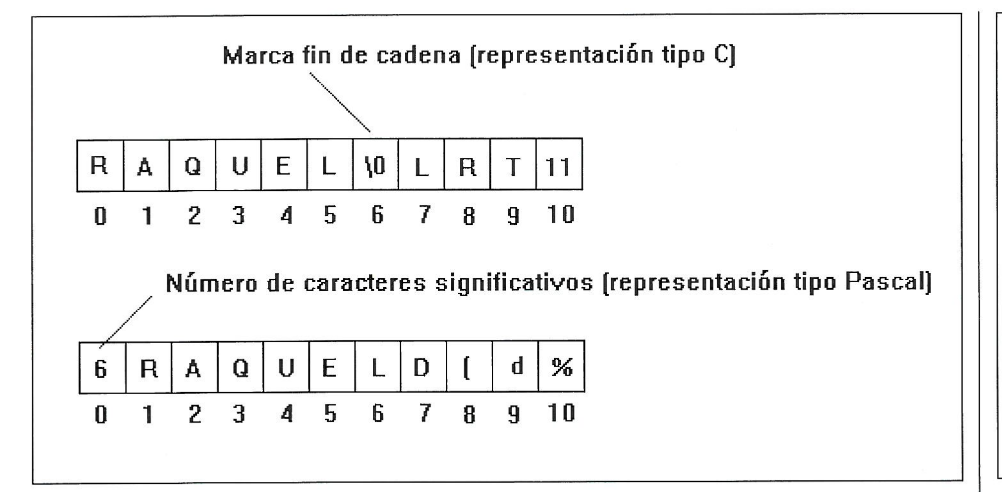
Un literal numérico se delimita entre dobles comillas o apóstrofo (dependiendo del lenguaje de programación), para diferenciarlos del resto de los elementos del lenguaje. Así, 'tablero', será un literal alfanumérico y representará lo que hay entre comillas, el estar limitado entre comillas lo hace distinguible dentro de un programa de las variables y de las instrucciones. Por ejemplo, si tablero fuese una variable numérica con valor 12, las instrucciones *escribir (tablero)*, daría como resultado en pantalla el número 12 (es decir el contenido de la variable), y *escribir ('tablero')*, que daría como resultado en pantalla *tablero*, son diferentes.

Existen en los lenguajes de programación la posibilidad de declarar variables capaces de almacenar cadenas de caracteres (este almacenamiento en memoria se realiza en posiciones contiguas). Una de las características que interesa a la hora de almacenar cadenas de caracteres es su longitud, así se definirá longitud de una cadena como el número de caracteres que ésta posee. Existe la cadena vacía, que es la cadena que no contiene ningún carácter (su longitud es 0), ésta no debe confundirse con una cadena que contenga todos sus caracteres blancos (es diferente la cadena vacía '', que la cadena '   ' que contiene tres caracteres blancos).

VARIABLES DE TIPO CADENA

En todos los lenguajes de programación las variables de tipo cadena basan su representación en *arrays* de caracteres, o incluso son simples *arrays* de caracteres con una serie de convenciones propias que permiten utilizar una serie de funciones de librerías construidas para el tratamiento de estas cadenas.

En la definición original del lenguaje de programación Pascal, las cadenas




```
funcion palindroma ( cad: string): entero
|
  inf, sup: entero
  bol: logico
inicio
  sup = longitud (cad)
  bol = verdadero
  inf = 1
  mientras (bol) y (inf < =sup)
      si cad[inf] = cad[sup]
        entonces
          inf = inf + 1
          sup = sup - 1
        en otro caso
          bol = falso
  fin mientras
  palindroma = bol
fin
```

de caracteres no existen, por lo que inicialmente el almacenamiento y procesamiento de las cadenas debía realizarse mediante *arrays* y funciones definidas por el usuario. El compilador de Turbo Pascal (Borland) sí que incluye dentro de su repertorio de tipos básicos, un tipo para este fin, el tipo string (este caso es similar al de QBasic). Para declarar una variable de tipo string, solamente hace falta asignarle un identificador y a continuación especificar el tamaño máximo, entre corchetes, de la cadena que podrá albergar la variable, o bien no especificar el tamaño, en cuyo caso podrá almacenar una cadena de hasta 256 caracteres. Por ejemplo:

...
var
cadena_fij: string[12];
cadena_var: string;
...

La representación interna es como sigue. La primera posición del string contiene la longitud significativa de la cadena almacenada con un valor tipo word (un octeto, cuyo valor puede estar entre 0 y 255). Utilizando las funciones estándar para strings, el programador no necesita acceder a este valor, las funciones y procedimientos incluidas con el compilador se encargarán de esto. Por ejemplo, si se realiza la asignación *cadena_fij:= 'Raquel'*, el string seguirá ocupando 12 bytes, pero solamente serán significativos, en cuanto a su contenido, los 6 primeros. Cualquier operación realizada mediante procedimientos y funciones estándar sobre el string sólo afectará a tales posiciones de memoria, ignorándose el resto. Por ejemplo, si se realiza la instrucción *writeln(cadena_fig)*, solamente aparecerán en pantalla los 6 primeros caracteres que son los significativos. El programador puede decidir construir su propia representación de strings, pero el costo es muy alto, deberá construir funciones y procedimientos adecuados a tal representación, por lo que se aconseja que a menos que sea absolutamente necesario se utilice la representación del propio lenguaje.

En C, los strings como tales no existen, son simplemente *arrays* de tipo carácter definidos como ya sabemos, ahora bien, este lenguaje dispone de una biblioteca estándar llamada stdio.h, que contiene, entre otras, funciones básicas de entrada/salida que utilizan una serie de convenios para tratar a los *arrays* de caracteres. El principal convenio, es incluir el carácter '\0' (que representa el carácter con valor cero) en la siguiente posición del *array* donde acaba la cadena de caracteres, las funciones básicas de entrada y salida, como son respectivamente *scanf* y *printf,* observan esta regla. De nuevo el programador es el máximo responsable al aceptar o no este convenio, en caso de no aceptarlo tendrá que escribir sus propias funciones de entrada y salida, en caso de aceptarlo tendrá que tener en cuenta el detalle de la representación para construir posteriores funciones.

En el cuadro 3 se puede observar la diferencia, mediante un esquema, entre las anteriores representaciones.

OPERACIONES

Las operaciones con las variables de tipo cadena dependen del lenguaje y de las bibliotecas de que se disponga, existen, sin embargo, una serie de operaciones comunes a todos los lenguajes que son: operaciones de entrada/salida, comparación de cadenas (según el orden del código de caracteres utilizado), y función para calcular la longitud de una cadena. A partir de estas operaciones es muy sencillo construir cualquier otra función o procedimiento, como por ejemplo, comprobar si una cadena está incluida en otra, eliminación de caracteres en blanco al principio de la cadena, conversión de una cadena alfanumérica a numérica etc.

Ampliaremos el seudocódigo para poder incluir variables de tipo cadena, permitiendo la declaración de este tipo de variables haciendo seguir al identificador y tras dos puntos la palabra reservada string. Por ejemplo:

cadena: string;

En el cuadro 4 se muestra el seudocódigo para la función *palíndroma*, esta función devolverá el valor verdadero cuando la cadena sea palíndroma (cadena que leída de izquierda a derecha, coincide con su lectura de derecha a izquierda). Se supone que disponemos de una función que dada una cadena nos devuelve su longitud en formato entero, la cual se denominará *longitud*.

LAS SHELLS (II)

Fernando J. Echevarrieta

En el artículo anterior se estudió el bucle de shell como intérprete interactivo de comandos. En el presente, se procederá a la descripción de las transformaciones que la shell realiza sobre su entrada y a la explicación de las estructuras de control de programación.

Una vez conocidos los mecanismos de ejecución interactiva de las shells, así como los de configuración y ejecución, únicamente resta adquirir los conocimientos necesarios para la ejecución no interactiva, es decir, la realización de programas en shell. En esta entrega se presentará en primer lugar una serie de transformaciones que realiza el parser de la shell sobre su entrada, que dotarán de gran potencia a la ejecución de comandos y, posteriormente se mostrarán las estructuras de control de la shell. Estas estructuras son extremadamente sencillas ya que la complejidad de las shells reside en lo expuesto en el artículo anterior y en la primera parte de este, por lo que este mes, el lector debe ser capaz de realizar sus primeros scripts de configuración en shell, modificar los del sistema y realizar otra clase de programas que se le ocurran.

La shell realiza 5 tipos de transformaciones sobre la entrada antes de emprender ninguna acción

EL PARSER: PROCESO DE LA ENTRADA

Durante el estado correspondiente al proceso de la entrada, la shell procesa la línea de comando realizando una serie de transformaciones sobre ella, que darán lugar a las instrucciones que realmente se llevarán a cabo en la etapa de inicio de operaciones. Estas transformaciones pueden ser de cinco tipos, aunque no todos son contemplados en las diferentes shells. Sustituciones de history, 2. Alias, 3. Variables, 4. Sustituciones de comando y 5. Expansión de ficheros. Las tres últimas supondrán mecanismos fundamentales a la hora de realizar programas en shell. En general, salvo las sustituciones de history y expansión de ficheros, estos mecanismos no se suelen usar en profundidad de forma interactiva si no a través de los ficheros de configuración de la shell estudiados en el pasado número o como parte del control en la programación.

SUSTITUCIONES DE HISTORY

Cada comando o línea que la shell procesa (en adelante evento), sea o no con éxito, se suele almacenar en memoria en una history list, lista que es volcada a un fichero, generalmente .history cuando la shell termina (muere) con normalidad. El control de esta característica es fuertemente dependiente de shell, por lo que para profundizar lo mejor es acudir al manual en línea de la shell correspondiente. En general se puede llevar a cabo al menos mediante dos variables: una de valor que indica el número de eventos que deben ser almacenados (ej. set history = número en csh y HISTSIZE=número en bash) y una booleana que activa o desactiva la función (ej. (un)set savehist en csh o HISTFILE en bash). La mejor forma de averiguar qué variables hay que alterar

TABLA 1	
!	Activación sustitución history.
!n	Evento con número de secuencia n.
!-n	n-ésimo evento anterior.
!!	Último evento (lo mismo que !-1).
!	prefijo Más reciente comenzando por prefijo.
^xx^yy	Cambia xx por yy en el último
!*	Todos los argumentos del último evento.
!$	Último argumento del último evento.
!^	Primer argumento del último evento.
!:n	n-ésimo argumento del último evento.
evento:s/xx/yy/	Cambia xx por yy en el evento indicado

Sustituciones de history más frecuentes.

ÚLTIMO EVENTO	TECLEADO	INTERPRETADO
diff file1 file2 > file3	vi !$	vi file3
ls -l name	^-l^-ld	ls -ld name
srot file	^ro^or	sort file
vi file	cd dir; !vi	cd dir; vi file
vi file	cd dir; !-1	cd dir; vi file
setenv TERM=vt100	^=^	setenv TERM vt100
ls dir	^s^s -F	ls -F dir

Ejemplos de sustituciones de history.

es ejecutar el comando set sin parámetros para observar los valores por defecto.

Con este mecanismo, se pretende cubrir dos objetivos: por una parte, ahorrar trabajo evitando teclear más de lo necesario y, por otra, recuperarse ante fallos en línea de comando.

La activación de la función de transformación de history se realiza cuando la shell encuentra el signo de admiración (!) o el acento circunflejo (^) en la línea de comando. Estos signos supone una secuencia de escape que provocará la sustitución de la línea tecleada por otra, que será la que realmente se procesará. Después de la sustitución, el comando sustituto se incorporará a la history list como último evento. En la tabla 1 aparece la lista de las sustituciones más frecuentes y en la tabla 2 aparecen numerosos ejemplos.

Algunas shells permiten acceder de forma interactiva a los comandos anteriores de uno en uno para poder modificarlos. Así, la ksh, realiza esta función cada vez que se teclea CTRL+K y la bash, permite un desplazamiento por la history list mediante las teclas del cursor (para más información consúltese el libro "LINUX UNIX PARA TODOS" que se regaló con la revista). Sin embargo, en ocasiones no se podrán utilizar estas facilidades.

SUSTITUCIONES POR ALIAS

Las sustituciones por alias permiten identificar una secuencia de comandos por un nombre o alias elegido. La asignación en csh se realiza mediante el comando

alias [nombre [definición]]

y en bash mediante

alias [nombre[='definición']]

Si se emplea sin argumentos, mostrará una tabla con todas las definiciones de alias presentes en la shell. Si únicamente se indica el nombre, mostrará, si existe, la definición del mismo y, si se emplea de forma completa, creará la definición. Para deshacer definiciones se puede emplear unalias nombre.

Ejemplos útiles en csh (se deja como ejercicio al lector que los adapte a bash) pueden ser alias ^L clear (bastará teclear CTRL+L para borrar la pantalla); alias ls ls -F o alias ls 'ls -F' (en csh también se puede utilizar la comilla para agrupar y prevenir bucles de sustitución); alias cd.. cd .. (cd.. es un nombre válido de fichero en UNIX, que no será encontrado, por lo que para usuarios de DOS que no suelan escribir el espacio en blanco, éste es un ejemplo útil); o alias dir ls.

Respecto al uso de esta sustitución hay que destacar dos reglas fundamentales:

1. Sólo sirve para la primera palabra del comando (el nombre) 2. En la fami-

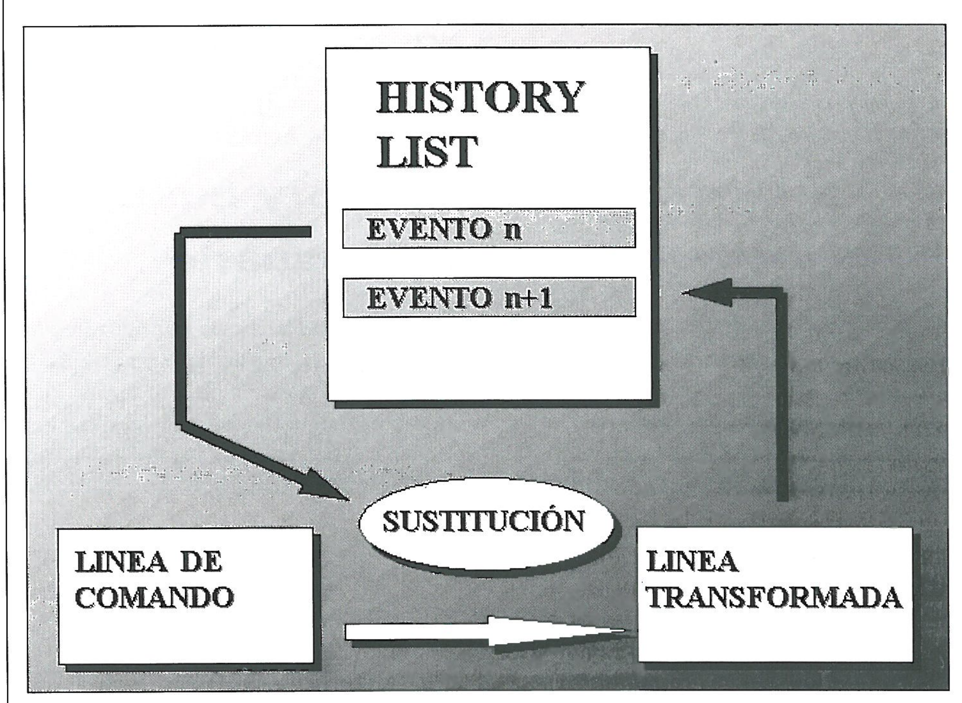


Figura 2. Sustitución de history.

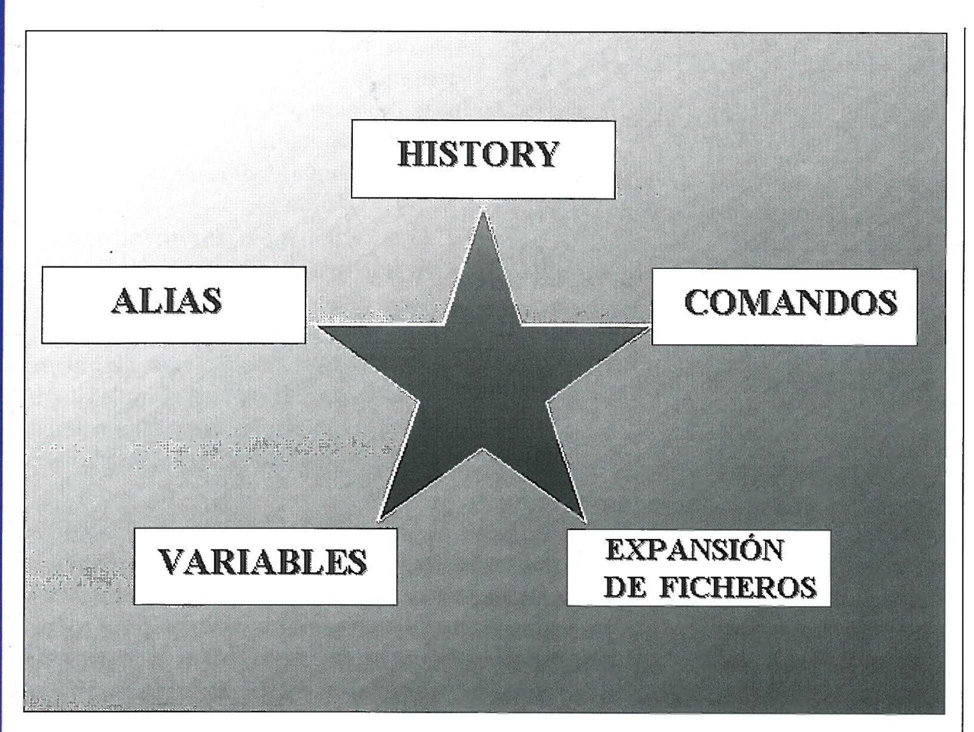


Figura 1. Transformaciones de la shell.

lia csh, en el alias se puede incluir una sustitución de history considerando:

a) La shell considera el propio comando como último evento y no el que realmente fue último y

b) Se sustituye el comando completo. Esto proporciona una gran potencia. Así, en csh

alias cd 'cd\!*;pwd'

equivale a decir "Oye shell, estate atenta y cuando yo teclee cd algo, lo que quiero que hagas es ejecutar cd algo; pwd". Como se ha visto, !* equivale a todos los argumentos del último evento. Si no se cumpliera la regla 2a, esta sustitución no tendría sentido. En shells como la bash que no admiten esta sustitución, se puede realizar definiendo una función de shell que se definirá más adelante. Otra regla que se observa en el ejemplo es la necesidad de incluir el carácter de escape en la definición del alias, sin el cual, la shell sustituiría el valor de !* en el momento de la definición, con lo que tomaría un valor fijo. De esta forma, no se interpreta !* durante la definición del alias sino cada vez que se teclea cd.

La sustitución de history busca ahorrar trabajo al usuario

SUSTITUCIÓN DE VARIABLES

Como se ha venido adelantando desde el capítulo 2 del curso, todas las shells tratan con variables que siempre son cadenas de 0 o más palabras. Entre ellas, se pueden distinguir dos tipos: a) especiales de shell (como noclobber, estudiada el pasado mes) y b) de usuario (útiles para la programación en shell). Los nombres de las mismas son identificadores que comienzan por un char al que le pueden seguir letras, dígitos o el carácter de subrayado. La shell trata de realizar una sustitución de variable cuando encuentra el signo dólar $.

El mes pasado ya se mencionó que las variables se tratan de distinto modo en las familias sh y csh:

1. Familia sh: La asignación se realiza mediante una igualdad: ej. DIRECTORIO=/usr/local. Las variables especiales tienen su nombre siempre en mayúsculas: ej. PATH.

2. Familia csh: La asignación se realiza mediante el comando set: ej. set DIRECTORIO=/usr/local. Las variables especiales tienen su nombre siempre en minúsculas: ej. path. Además, una variable puede ser definida como un ARRAY, ej. set path = ($HOME/cmd /usr/local/bin) y se puede acceder de forma independiente a los elementos del mismo: ej. echo $path[1] o set path[1] = /home/echeva

En cualquier shell, set empleado sin argumentos, muestra el valor de todas las variables.

La bash, a pesar de ser familia sh, ha incorporado casi todas las ventajas de la csh y con ellas muchas variables booleanas "minúsculas" de la misma, como el citado caso de noclobber. Para mantener compatibilidad, las ajusta mediante la opción -o de set (set -o noclobber).

Siguiendo los ejemplos, si se teclea cd $DIRECTORIO la shell al encontrar el signo $ buscaría una variable llamada DIRECTORIO y realizaría la sustitución. De este modo es importante recordar:

1. Al declarar la variable no se utiliza $
2. Para referirse al contenido de la variable es necesario utilizar $.

Otro ejemplo que serviría para mostrar el valor de la variable es echo $DIRECTORIO.

VARIABLES DE ENTORNO

Para hacer que una variable sea conocida, no únicamente en la shell actual sino por todos lo procesos que nazcan de ésta, es necesario exportarla. Así pues, se podría hablar de variables de shell (específicas de la shell actual), como HOME, PATH o TERM (tipo de terminal) y variables de entorno (utilizadas por la shell actual y todos los procesos a partir de ella creados). Así, si una variable como TERM no se exporta, cualquier subshell o shell hija

TABLA 3	
*	Cualquier cadena de longitud 0 o más
?	Cualquier carácter
[..]	Cualquier carácter del rango Ej a[abc], capitulo[0-9]
~	Dir. home del usuario
~	userid Dir. home del usuario userid
{...,...}	Una de las posibilidades indicadas
[!..]	Los caracteres que no están en el rango (sólo en sh)

Expansión de ficheros.

no sabrá en que tipo de terminal se encuentra y lo mismo ocurrirá con cualquier proceso generado (recuérdese el mecanismo de ejecución de procesos a partir de shell descrito en el número anterior).

La sustitución de history es fuertemente dependiente de shell

1. Familia sh. Una variable de entorno se genera en dos pasos: a) Asignación de valor, ej. TERM=vt100

   b) Exportación de la variable, ej. export TERM.

2. Familia csh. Las dos acciones se pueden realizar en un sólo paso: setenv TERM vt100.

Nuevamente, la bash es un caso especial y además del mecanismo sh estándar admite una exportación en un paso: export TERM=vt100

SUSTITUCIÓN DE COMANDOS

La sustitución de comandos es el mecanismo más simple de la shell y uno de los más potentes. Cuando una shell encuentra unas comillas invertidas, procede a la sustitución de lo que engloban por el resultado de procesar lo que engloban. Por poner un ejemplo, set dir=`pwd` o dir=`pwd` (según shell), asignaría a la variable dir el nombre del directorio por defecto (resultado de ejecutar pwd).

Por supuesto, entre las comillas se puede encerrar cualquier cosa, combinando toda la potencia de lo ya expuesto.

TABLA 4	
$#	Número de argumentos (parámetros)
$*	Todos los argumentos de la shell
$	Opciones pasadas a la shell
$?	Valor del último comando
$$	PID de la shell
$!	PID del último comando &
$HOME	Directorio home del usuario
$IFS	Delimitador de palabras en argumentos
$PATH	Camino de búsqueda de ficheros
$PS1	Prompt de 'espera' (habitual)
$PS2	Prompt de 'continúa'

Variables más usadas en sh scripts.

EXPANSIÓN DE NOMBRES DE FICHEROS

Si uno reflexiona se dará cuenta de que la mayoría de las palabras que se utilizan en UNIX son nombres de ficheros. Así, no podía faltar una sustitución para ello. Ésta se realiza a través del uso de patrones (muy similares a los del DOS). Así, es posible teclear ls -l *.c, ls -l dir/*.c o ls -l book/section*/*.ms.

En cualquier caso se recuerda que UNIX no existen extensiones de fichero, así que *.* no aporta más significado que *a*, simplemente el de cualquier nombre que contenga un punto o un carácter "a" respectivamente. Así, bastará utilizar * para referirse a todos los ficheros.

Aunque son muy similares no conviene confundir la expansión de ficheros en DOS con la de UNIX

La tabla 2 muestra las posibles sustituciones. El hecho de realizar la expansión de ficheros recibe en inglés el nombre de globbing. Así pues, existe una variable booleana csh (utilizable en bash con la opción -o de set) llamada nogglob que inhibe (activa) la expansión cuando se encuentra (des)activada. Nuevamente, el verdadero potencial de estas facilidades se encuentra en la programación en shell.

CARACTERES ESPECIALES: QUOTING.

En ocasiones es necesario evitar que se realice alguno de los mencionados procesamientos de la shell. Para ello es necesario recurrir a mecanismos de quoting (el autor prefiere no traducir el termino por "quotado") o de "cita literal". Para ello se utilizan tres caracteres: comilla simple ('), comilla doble (") y backslash (\). Se recuerda que la comilla invertida se utiliza para sustitución de comandos, con lo que todas las comillas en UNIX van dotadas de carga semántica.

El carácter \ como se mencionaba anteriormente, es un símbolo de escape que indica a la shell que no debe interpretar el carácter inmediatamente posterior.

Un ejemplo de esto puede surgir de la necesidad de referirse a un fichero llamado h&p.txt. La shell, al encontrar el ampersand & consideraría que es un terminador paralelo (ver artículo del mes anterior), y produciría al menos, un mensaje de error por no poder ejecutar p.txt. La solución sería simple:cat h\&p.txt

Si lo que se necesita es "escapar" una cadena entera, se puede emplear la comilla simple ('), aunque las sustituciones provocadas por ! (history) sí se llevarán a cabo. Un ejemplo de ello puede ser echo 'a + b=c * d'. Si en cualquier momento hay que escribir una comilla simple, y lo que se desea es evitar que la shell la interprete, ya se conoce el remedio: ej. echo echeva\'s server.

El tercer y último mecanismo de quoting es la doble comilla, que funciona igual que la simple pero permite las sustituciones por $ (variables) y comilla invertida (comandos) en su interior: ej. echo "Este dir es $cwd" o set fecha="Este dir es `pwd`"

PROGRAMACIÓN EN SHELL

Aunque en la tabla 6 se han representado las analogías entre las estructuras de programación de la familia sh y csh, se presentarán las estructuras de la sh, que por su mayor simplicidad se han convertido en el estándar de facto en la realización de shell scripts UNIX.

FORMATO DE UN FICHERO DE TEXTO EJECUTABLE

Cuando una shell comienza la ejecución de un comando externo, lee los primeros caracteres del fichero ejecutable para determinar de qué clase es. Si comienza por un número mágico (un código prácticamente imposible de generar en un fichero de texto) asume que el fichero es binario. En caso contrario, supone que se trata de un fichero de texto ejecutable. Si los dos primeros caracteres del fichero son !# seguidos inmediatamente del nombre de un fichero ejecutable será este fichero el que se lanzará para interpretar el fichero de texto. Así pues, si se programa en shell conviene comenzar todos los scripts con la línea:

#!/bin/sh

TABLA 5

*, ?	Símbolos comodín
[ccc]	Cualquier carácter del conjunto.
[a-z0-9]	Cualquier carácter de los rangos
"aaaa"	Exactamente aaaa
a|b	a ó b

Patrones de la instrucción case

TABLA 6

sh	csh
for var in lista	foreach var (lista)
do lista_de_comandos	lista de comandos
done	end
if lista_de_comandos	if (expr1) then
then lista_de_comandos	lista_de_comandos
[else lista_de_comandos]	[else if (expr2) then
fi	lista_de_comandos]
	[else lista_de_comandos]
	endif
case var in	switch (str)
patron) lista_de_comandos	case patron:
;;	breaksw
[*) lista_de_comandos	...
;;]	[default:
esac	lista_de_comandos]
	endsw
while lista_de_comandos	while (expr)
do lista_de_comandos	lista_de_comandos
done	end
var = string	set var = string
until	no disponible
no disponible	goto etiqueta
break [n]	break
continue [n]	continue
no disponible	set var = (s1, s2 ...)
no disponible	set var[n] = string
var = `expr`	@var = expr
no disponible	@var[n] = expr
no tiene expr. logicas internas	tiene expresiones lógicas internas
comando /bin/test	evaluación interna status ficheros

Analogías entre las estructuras de csh y sh.

ESTRUCTURAS DE CONTROL

La tabla 4 representa las variables más utilizadas en la realización de sh scripts. Existen otras variables llamadas posicionales que hacen referencia al argumento n-ésimo que se le pasa a la shell y se designan por $n: $0, $1, $2... Equivalen a las variables %1, %2, %3 etc, de los ficheros batch del DOS. El prompt de "continúa" es el que aparece cuando se utiliza una estructura de control de programa desde línea de comando, es decir, en forma interactiva. Recuérdese que al tratarse de un lenguaje interpretado, se puede utilizar desde línea de comando, como ocurría con el BASIC de aquellos entrañables ordenadores domésticos...

La bash permite moverse por la history list mediante los cursores

Instrucción exit

Provoca la salida del programa con el valor de retorno indicado.

exit num

Instrucción CASE

```
case var in patron1) comandos;;
            patron2) comandos;;
esac
```

Provoca la ejecución de una o otra serie de comandos condicionada a que el valor de la variable var satisfaga al patrón i. Los patrones de case figuran en la tabla 5.

Un ejemplo:

```
echo -n 'te vas?'
read respuesta
case $respuesta in      [sS]*) echo 'adios' ;;
[nN]*) echo 'seguimos' ;;
   *) echo 'respuesta no válida';;
esac
```

En el presente ejemplo se observa el mecanismo de lectura de variables desde la entrada estándar a través de la instrucción read.

Instrucción if

Realiza una ejecución condicional. Obsérvese que la condición es el valor de retorno de un comando, TRUE (valor 0) o FALSE (cualquier otro valor). Por lo tanto, el comando también será ejecutado. Hay que hacer

notar que, en ocasiones, el efecto producido no es el deseado ya que un comando puede indicar TRUE aunque no haya realizado su objetivo. En caso de no observar ejecuciones como las previstas es necesario consultar el manual del comando para conocer sus valores de retorno.

```
if cmd1 then cmd2 fi
if cmd1 then cmd2 else cmd3 fi
```

Ejemplo:

```
if
grep $1 /etc/passwd > /dev/null
then
echo $1 encontrado
else
echo $1 no encontrado
fi
```

Instrucción for

Ejecuta un bucle controlado por la lista de variables dada. Suele emplearse casi siempre asociado a expansión de ficheros. Es importante escribir cada instrucción en una línea tal y como se indica a continuación.

```
for var in listvar
do
comandos
done
```

Ejemplo:

```
for i in $*
do
echo $i
wc $i
done
```

Instrucción until

Representa el clásico bucle estructurado de ejecución de la serie de comandos hasta que comando devuelva un valor 0 (TRUE). Análogamente a for, la condición es un comando

until comando do comandos done

Ejemplo:

```
until who | grep maria
do sleep 60
done
```

Ejecuta sleep 60 hasta que el programa who encuentre a María conectada al sistema (la instrucción sleep se explicó en el capítulo dedicado a X WINDOW). Una vez María se conecte el bucle termina y el programa continúa.

Funciones de shell

La programación en shell es estructurada. Se pueden definir nuevos comandos de la shell a través de funciones, que se definen del mismo modo que los procedimientos en lenguaje C. Las funciones, al contrario que las variables, no son exportables.

```
nombre()
{
        comandos;
}
```

En un próximo número se realizará un repaso de todo lo expuesto hasta ahora analizando distintos programas escritos en shell así como algunos ficheros de configuración. Mirando más a largo plazo comentar que la serie de artículos de Unix dará un giro radical para comenzar a tratar sobre una de las tecnologías más de "moda": World Wide Web. Dado que la mayoría de los medios de comunicación hablan sobre el sistema, pero todos cuentan lo mismo nosotros seremos los primeros en profundizar en este tema. Seguro que este tema será de interés dado que la implantación del WWW e Internet en España ha creado demanda de un nuevo puesto de trabajo. Otros asuntos que también pasaremos a estudiar serán los fundamentos de HTPP, y el lenguaje HTML.

Entre tanto, el autor puede ser localizado mediante correo electrónico a la redacción o a través de correo electrónico internet (echeva@dit.upm.es) o compuserve (100646,2456) para resolver cualquier duda que se presente.

En la shell se encuentran las conocidas estructuras: exit, case, for, until y while

TRATAMIENTO DIGITAL DE IMÁGENES (VI)

Juan Ramón Lehmann

Antes siquiera de modificar una imagen, lo primero que se debe hacer es obtenerla. La obtención de una imagen, se puede lograr de muchos y variados modos. La forma más común de obtenerlas es a través de un scanner o de una tarjeta digitalizadora de vídeo.

Normalmente la salida del scanner es una imagen en RGB de longitud fácilmente calculable en función de la profundidad del color y la resolución de la misma. Habitualmente los programas de manejo de scanner, producen una salida en el formato de imagen que se elija, por lo que la conversión de RGB no es necesaria. De todas formas el conocimiento de los sistemas de color (tales como RGB, HLS, CMYK) es sumamente importante como ya vimos en el número 11 de esta revista.

En el caso de las digitalizadoras de vídeo, usan otra codificación (YUV 4:1:1, YUV 4:2:2), y aunque la mayoría disponen de drivers para grabar en el formato deseado, algunos modelos antiguos no los tienen (como la PIB 1024). Las tarjetas digitalizadoras de vídeo como la VideoBlaster (Creative Labs), la VideoMovie (de Fast Ibérica) y otras tantas usan todas el mismo tipo de codificación, lo cual las hace sumamente compatibles, (a nivel software) a la hora de convertir las imágenes desde su buffer propio a un formato más estándar (por ejemplo RGB).

Este tipo de codificación se verá más adelante, con carácter informativo.

Dependiendo del dispositivo de captura, dispondremos de una imagen con un número determinado de colores. Este número de colores (16 millones, 64000 colores, 32000 colores, 256 colores, etc.), muchas veces hemos de reducirlo (o cuantizarlo), con objeto de economizar en espacio de almacenamiento. Esta técnica que se la denomina cuantización, fue explicada en el número 4 de esta misma revista.

El sistema de codificado de color YUV se suele utilizar en la mayoría de las tarjetas digitalizadoras de vídeo

En este artículo se hará una recopilación de todos los capítulos hasta ahora publicados con objeto de sintetizar y a su vez añadir algunos puntos que pueden ser de interés.

Una vez realizados estos procesos, disponemos de una imagen (en un formato X genérico) para poder manipularla a nuestro antojo.

EL SISTEMA YUV

El sistema de codificado de color YUV se suele utilizar en la mayoría de las tarjetas digitalizadoras de vídeo. El sistema YUV es análogo al PAL. Este sistema separa la luminosidad del color. Son métodos de almacenamiento lineal, por lo que son bastante usados en sistemas de procesamiento de imágenes o compresión de éstas.

El sistema se codifica de la siguiente manera:

Y=luminosidad





U,V= valores para azul y rojo respectivamente.

La tabla de conversión a RGB sería:

Y=	0.301*R	+	0.586*G	+	0.113*B
U=B-Y =	-0.301*R	-	0.586*G	-	0.887*B
V=R-Y=	0.299*R	-	0.588*G	-	0.113*B

Los valores en la anterior tabla de conversión son siempre en 7 bits. Los valores U y V son enteros con signo, escalados por 127/226 y 127/179 respectivamente, con objeto de codificarlo en 7 bits.

- Codificación YUV 4:1:1
Es una implementación de la norma anterior. Se disponen en 4 muestras de Y por cada muestra de U y V. Mediante esta técnica se le da más importancia a la luminosidad que al color, lo cual se ajusta más a las imágenes reales de vídeo. Se incluye un listado de conversión de formato YUV 4:1:1 a RGB para mejor comprensión del sistema de conversión.

El código fuente leerá dentro de un buffer una <<línea>> en formato YUV, y la convertirá en formato RGB

Los tipos de formatos gráficos suelen disponer de algoritmos compresores

MANIPULADO DE IMÁGENES
Una vez disponemos de una imagen, podemos desear por diversos motivos:

Restaurarla. Debido a perdida de información.
Mejorarla. Debido a la mala calidad de la misma.
Retocarla. Con objeto de ampliarla, o simplemente rotarla.
Segmentarla. Debido a necesidades del usuario. Por ejemplo, el OCR separa el texto del fondo para su posterior análisis)
Analizarla.
Comprimirla/Codificarla.

RESTAURACIÓN
Básicamente la restauración de imágenes trata de eliminar la distorsión (o ruido) que la imagen original sufre. Con este propósito, se utilizan filtros tratando de minimizar el error local o buscando la máxima entropía. Para conseguir un resultado óptimo, deberemos conocer la fuente de esa distorsión, y en base a ella utilizar el filtro adecuado. En el número 9 de Sólo Programadores se vieron varias técnicas que funcionan muy bien a la horas de realizar funciones restauradoras.

La imagen a superponer, debe tener un patrón binario que indique las zonas a tratar y las zonas a ignorar

- Filtros de suavizado.
- Método de la media restringida.
- Filtrado de la Mediana.
- Filtrado homomórfico (Nº 10 Solo Programadores)

MEJORA
En muchas ocasiones, cuando se dispone de imágenes con poca calidad, (por ejemplo con poco contraste) no se pueden visualizar correctamente (los bordes poco detallados confunden las imágenes de fondo con el objeto focal). Éste y otros motivos nos pueden llevar a desear mejorar la imagen. Técnicas para la mejora de la imagen están recogidas en el número 10 de Sólo Programadores.

- mediana laplaciana
- unsharp masking

RETOQUE
Podemos desear retocar la imagen para aumentar una porción de ella (zoom), girarla para dar una impresión de ángulo, desplazarla, o simplemente



superponerla (blending), a otra imagen. Todas estas técnicas (excepto la de superposicionado, que veremos posterioriormente), fueron recogidas en el número 12 de Sólo Programadores.

SUPERPOSICIÓN

Básicamente se trata de superponer 2 imágenes de acuerdo con un criterio de transparencia. Se ha de disponer de una imagen base (o la imagen que va a estar en background) y una imagen a superponer. La imagen a superponer, debe tener un patrón binario que indique las zonas a tratar y las zonas a ignorar. En este ejemplo (Fig. 1a) se ha utilizado como discriminador las zonas blancas de la Figura 1. Una vez disponemos de dichas imágenes, se mezclan punto por punto (todos sus componentes RGB) hasta producir el resultado deseado (Figura demo).

El modo de mezclar los componentes variará según el algoritmo utilizado. El algoritmo aquí utilizado fue:

Oxy=(I2xy*Alpha+(1-Alpha)) or I1xy
en el cual Oxy, es un punto en la imagen de salida. I2xy es un punto en la imagen de corte (la imagen que actúa de patrón binario, discriminada por <255). Alpha es el factor de transparencia (en el ejemplo se usó un .4, siendo sus limites un valor entre 0 y 1)y I1xy es un punto en la imagen de background.

En el ejemplo (Fig. 1a) se han utilizado dos imágenes en formato TGA grabados a 320*200 a 16 millones de colores (true color). Las imágenes han sido cuantizadas usando el algoritmo del árbol octal y mezcladas según la ecuación anteriormente explicada. El TGA resultante, fue convertido a GIF para la inclusión en la revista.

Para la restauración, se utilizan filtros tratando de minimizar el error local o buscando la máxima entropía

SEGMENTACIÓN

Este tipo de filtro es muy útil si deseamos separar partes de la imagen, que deseamos analizar con posterioridad. La binarización (Sólo Programadores nº 9) es un procedimiento muy utilizado para tratamientos de OCR, obteniendo las imágenes del texto por el contraste con el fondo blanco (habitualmente).

ANÁLISIS

El análisis es posterior a la segmentación. Dentro del análisis, entra la tarea de reconocer objetos, identificarlos y procesar los mismos, entrando de lleno en el reconocimiento digital de imágenes.

BIBLIOGRAFÍA

La literatura al respecto es amplia y variada. Aunque es difícil encontrarla en castellano, los títulos disponibles aumentan día a día.

COMPRESIÓN/CODIFICACIÓN

Una vez se dispone de una imagen en el framebuffer, habitualmente se comprime antes, o al grabar la misma en un formato determinado. Los tipos de formatos gráficos suelen disponer de algoritmos compresores, basándose gran parte de los mismos en el algoritmo de Huffmann o alguna de sus variantes. El motivo de comprimir las imágenes, como se puede ver fácilmente, tiene como objetivo el reducir al máximo el espacio utilizado en disco por dicha imagen.

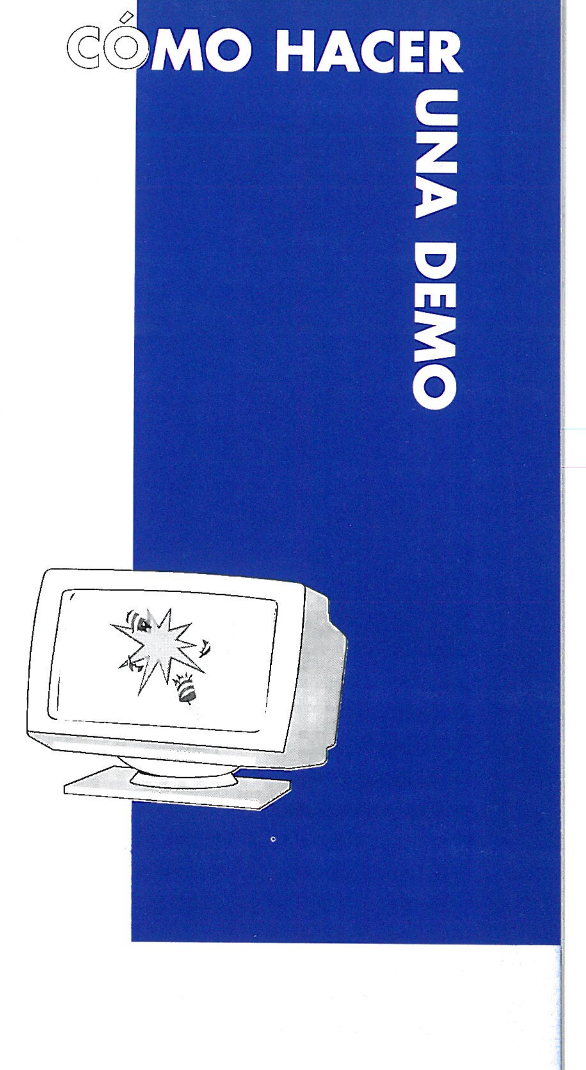
La codificación de imágenes se realiza con objeto de trasferir imágenes de forma segura. Esto es, encriptación de imágenes. Los métodos de encriptación son muy variados, desde el burdo XOR (que calcula el complemento) hasta la utilización de algoritmos con clave pública, más complejos y seguros, pero que cobran su seguridad a costa del tiempo de codificación.

En Castellano:
Tratamiento digital de imágenes. (Alberto Domingo Ajenjo, Anaya Editorial)

En Inglés:
Graphics Gems I (Glassner, Academic Press)
Graphics Gems II (Arvo, Academic Press)
Graphics Gems III (Kirk, Academic Press)
Graphics Gems IV (Heckbert, Academic Press)
Practical image processing (Craig A. Lindley)
ZEN of Graphics Programming (Michael Abrash)

ASSEMBLY 95: LA GRAN "PARTY" DEL PC

Luis Crespo



Como cada verano desde 1992, se celebra en Finlandia el Assembly, la party de PC más famosa del mundo. Este año, en su cuarta edición, ha destacado la buena organización del evento.

El Assembly es básicamente un concurso de demos. Pero en realidad es mucho más. Según los organizadores es la mayor party del PC del mundo, y seguramente tienen razón, ya que en lo que se refiere a afluencia de público sólo es superada por The Party en Dinamarca, dedicada al mundo del Amiga.

¿QUÉ ES UNA PARTY?

Las parties empezaron en el ámbito de los ordenadores Amiga, siendo una reunión de amigos que, unidos por la afición a los ordenadores, se juntaban para intercambiar programas, ficheros, ideas, etc. Al mismo tiempo se iban haciendo más famosas las demos. Aquellos programas cuya finalidad principal es la de impresionar a quien los ve, demostrando el profundo conocimiento de la máquina que tiene el programador. De hecho las primeras demos aparecieron en el mundillo de la piratería, en donde los grupos de *crackers* (rompedores o desprotectores de programas) demostraban de lo que eran capaces con la máquina. De modo que en las parties se empezaron a intercambiar demos, y se empezaron a hacer concursos. Y ahí empezó todo.

LOS CONCURSOS

En toda party de hoy en día hay varios concursos, no sólo de demos. Los concursos de demos, además, se clasifican según categorías, en función del tamaño en Kb de los ficheros. Así, en el Assembly 95 ha habido tres categorías para demos de PC: hasta 4 Mb, llamadas propiamente demos, o a veces mega-demos; hasta 64 Kb, llamadas intros; y hasta 4 Kb, llamadas 4 Kb intros. En el entorno Amiga (menos importante en el Assembly, ya que está orientado al PC), las demos pueden ocupar hasta 4 disquetes y las intros hasta 40 Kb. Y también, más bien por cuestiones nostálgicas, se sigue conservando el concurso de demos de Commodore 64.

Las otras dos categorías clásicas de una party son la de gráficos y la de música. En el Assembly el concurso de música se divide en canciones de 4 canales, y canciones de hasta 32 canales, que pueden ser módulos de cualquier tipo (MOD, S3M, XM, etc, siempre y cuando se suministre el reproductor adecuado), o ficheros MID. En cuanto a gráficos, se hace distinción entre dibujos hechos "a mano" con un programa de dibujo, y escenas renderizadas por un programa de 3D.

Este año se ha introducido una nueva categoría que ha tenido bastante éxito, y que promete mucho en el futuro: las animaciones. En las animaciones, en contraposición a las demos, cada imagen ha sido calculada previamente con un programa de 3D, o ha sido dibujada por el artista con la ayuda de un programa de animación. Además la animación puede ir acompañada de una banda sonora (también "precalculada") que le añade espectacularidad.

En el Assembly, los ganadores de cada categoría se deciden por "votación popular": junto con la entrada al recinto se le entrega a todo el mundo un *voting disk*, en el que hay un programa con el que se votan las mejores producciones de cada categoría. Al acabar los concursos se devuelve el disquete a los organizadores y éstos los utilizan para hacer el recuento de votos, con los que se deciden los ganadores. Antes de presentar las producciones por la pantalla



A la derecha, Berti, uno de los programadores de Noon. Este joven grupo se impuso en la categoría de Demos en la última edición del certamen Assembly.

gigante, un grupo de gente formado por miembros de grupos famosos hace una preselección. De esta forma hay tiempo suficiente de mostrar las mejores, porque el número de obras que se presentan en ciertas categorías es realmente elevado. Por ejemplo, en la categoría de música se presentan cientos de módulos, y sólo se seleccionan 15 para presentarlos al público.

La presentación de las producciones preseleccionadas de cada categoría tiene lugar en orden creciente de importancia. Tras un primer día en el que no hubo ningún evento, en el segundo día se presentaron las canciones, gráficos y demos de Commodore 64. El tercer día tuvo lugar la competición de gráficos dibujados y renderizados, de música de 4 y 32 canales, de intros de 4 Kb, de intros de Amiga y de intros de PC. Por último, en el cuarto día se presentaron las demos de Amiga, las animaciones, y finalmente el evento más importante: las demos de PC.

EL LUGAR

Para poder celebrar una party es necesario un local bastante grande. El Assembly del 93 se celebró en el gimnasio del colegio de un pueblo, el del 94 en un estadio de hockey de Helsinki, y este año en un pabellón de una feria de muestras de Helsinki. En lo que se llama el party place, se colocan grandes mesas como si se tratara de un comedor de colegio, y se proporcionan enchufes suficientes para conectar todos los ordenadores y aparatos que traen los visitantes. En el centro del local hay una gran pantalla sobre la que se proyectan las creaciones que se presentan a concurso, se informa sobre los distintos eventos y se muestran producciones antiguas. Durante las proyecciones, se apagan todas las luces y la gente se coloca lo más cerca posible de la pantalla para poder disfrutar de las obras que se presentan a concurso. Este año, además, se ha habilitado un pabellón para dormir. La gente pudo colocar sus sacos de dormir e incluso montar alguna tienda de campaña, para las pocas horas de sueño de que se dispone en un evento de este tipo.

EL AMBIENTE

La mayoría de los asistentes al Assembly se traen su ordenador, más aún si viven cerca del lugar. Algunos también se traen todo el equipo de música, e incluso cafetera y microondas, montando todo un tinglado en su mesa. Es bastante normal que los grupos que presentan algo a concurso aprovechen las últimas horas de que disponen antes de la entrega (la famosa deadline), para terminar su trabajo. De hecho esto es parte de la gracia de las parties, aunque a veces apuran demasiado y no pueden dormir porque tienen que arreglar un bug que les ha salido en el último momento. A menudo las demos que se presentan no están del todo terminadas, y por ejemplo falta poner el típico scroller con los créditos y los greetings (saludos) a otros grupos. No es extraño encontrase a alguien dormido sobre el teclado, quién sabe si tras arduas horas de trabajo para poder presentar la producción a tiempo.

De la gente que acude a una party se pueden hacer una clasificación: en primer lugar hay que destacar los grupos de demos. Montan sus ordenadores en una mesa, y cuelgan una pancarta con el nombre del grupo. Algunas pancartas son realmente grandes y espectaculares; este año, la gente de EMF (autores de la demo ganadora del Assembly del 94) se trajo un proyector con el que mostraban su nombre en enormes letras luminosas en una de las paredes. Si el grupo trae una demo o intro, sus miembros suelen estar muy ocupados ultimando los detalles. La mayoría de los grupos van con su camiseta oficial, en la que aparece el logo del grupo y el nombre del miembro. Algunos incluso van más allá y visten un cierto uniforme. Por ejemplo, el año pasado los miembros de Prime (autores de la intro ganadora del Assembly del 94) iban vestidos con monos azules, y este año se podía ver rondando a los Byterapers (grupo de demos de Commodore 64) vestidos con largas gabardinas negras, o a unos desconocidos vestidos con batas blancas de científico.

También acude gente que quiere disfrutar del ambiente de la party y ver las producciones que se presentan, las novedades, etc. Además, cualquiera que sea aficionado a la programación de algo más que gestión, es decir, gráficos, sonido, ensamblador, optimización, acceso a los dispositivos de la máquina, hacking, etc, le puede sacar partido a una party. Ahí se puede ver lo último en técnicas de 3D y gráficos, nuevas utilidades de sonido, nuevas rutinas... Durante las parties, los coders (programadores) intercambian conocimientos, explicándose lo que ha descubierto cada uno para exprimir del procesador unos cuantos ciclos menos en una rutina. Tras una party en la que



aparece una demo con un nuevo efecto hasta entonces considerado imposible de realizar en tiempo real, los coders no paran de darle vueltas hasta que consiguen descubrir la técnica que se utiliza. Las demos, de hecho, siempre son una especie de banco de pruebas de lo que es posible realizar con un PC. Frecuentemente, en juegos comerciales se utilizan efectos que antes se han visto en demos.

No son pocos los que van al party porque les queda cerca y se traen sus ordenadores para pasarse el día jugando. Últimamente, con la fama que está adquiriendo el evento, también rondan periodistas, cámaras de televisión, e incluso caza-talentos. Estos pertenecen a empresas productoras de software recreativo, y buscan genios entre los miembros de los grupos más famosos, y tras el reparto de premios, de los grupos ganadores. Otros aprovechan el Assembly para presentar un nuevo programa o una nueva versión de un programa famoso, o para vender un CD-ROM que ellos han producido. Los patrocinadores del evento, a su vez, montan un stand en el que exponen sus productos y los venden a precios de oferta.

ESTE AÑO

Como novedad, en el Assembly 95 se ofrecía, además de los enchufes de corriente para conectar los ordenadores, conectores a la red del party: PartyNet. Si el ordenador estaba provisto de tarjeta de red, podía conectarse a una red con varios servidores Novell y acceso a Internet. De esta forma, la gente pudo coger y dejar las producciones más recientes sin siquiera levantarse de la mesa, o también jugar al Doom o a otros juegos en red. A través de Internet, se podía, como no, coger ficheros de cualquier parte del mundo vía ftp, entrar a máquinas remotas vía telnet, e incluso mantener conversaciones con cualquier otro usuario de Internet vía irc.

La mayoría de los asistentes se traen el ordenador

Por otra parte, éste ha sido sin duda el año de Phong. La técnica de Phong se utiliza en 3D para dar a los objetos una textura de brillos metalizados. El año pasado el grupo alemán Legend Design presentó la primera demo que utilizaba el sistema de Phong. A principios de año, el grupo Complex ganó "The Gathering" (una party celebrada en Noruega) con la demo Dope, que con una técnica nueva conseguía acelerar la velocidad de las imágenes tridimensionales con brillos metalizados. Tras conocerse esta nueva técnica, prácticamente todo el mundo se ha lanzado a codificar sus rutinas de 3D con Phong. Y como consecuencia de ello, casi todas las demos e intros presentaban más y más objetos tridimensionales con brillos metalizados. El objeto más utilizado para mostrar los brillos de Phong ha sido el toro (es decir, el donut), hasta el punto que ha faltado poco para coger una indigestión de fabulosos donuts brillantes rotando en la gran pantalla. Por su gran parecido técnico con el nuevo Phong trucado, también se han visto bastantes producciones con environment mapping, es decir, objetos que reflejan las imágenes que hay a su alrededor.

De entre los grupos que han asistido, cabe destacar la asistencia de dos grupos españoles: como cada Assembly desde su segunda edición, los ya veteranos Iguana (autores de la demo Heartquake que quedó tercera el Assembly del año pasado), este año no presentaron ninguna demo porque se reservan para la Euskal Party; y un nuevo grupo, Miracle, presentó su primera demo (más sobre esto en el siguiente párrafo). De fuera de España, entre otros, acudió el Cubic Team, un grupo alemán famoso por el reproductor de módulos Cubic Player, presentando una animación y una intro; el grupo finlandés Capacala, que por lo que parece se recuperó del mal trato que la organización le dio a su demo el año pasado (muy buena, por cierto), presentó una nueva demo; el grupo EMF no presentó nada este año; del famoso grupo Future Crew tan sólo rondaban algunos miembros desperdigados por el party; en cambio Orange y Halcyon, dos grupos muy activos recientemente, presentaron una demo y una intro respectivamente.

LAS PRODUCCIONES DE ESTE AÑO

Este año la demo ganadora ha sido "Stars : Wonders of the world", del grupo francés Nooon. La demo consiguió sobrecoger porque utilizaba el sombreado de Phong con una estética de gran calidad, y porque los objetos que se movían en pantalla eran de gran complejidad. En ella se puede ver una avispa dorada con más de 23000



caras, o un torso femenino bailando con un hula-hoop (ambas figuras por supuesto utilizando Phong), todo ello reflejándose en un aro de goma mediante environment mapping. También una especie de erizo multicolor y un muro que cambia de textura mientras una cara aterrorizada intenta atravesarlo sin conseguirlo, entre otros impresionantes efectos, hicieron a la demo merecedora del primer premio. El efecto de la cara y el muro, sin ir más lejos, se presentó hace pocos años en el Siggraph, un concurso para profesionales en el que se utilizan máquinas mucho más potentes.

Pero no todo fue puro 3D en el concurso de demos. La demo "Little Green Men", del grupo Kosmic, se salió del rutinario "más de lo mismo" de la mayor parte de demos de PC. Su demo no está dotada del último grito en efectos, pero tiene un tema, un hilo central: los pequeños hombres verdes venidos del espacio exterior. Esta demo podría ser perfectamente un video-clip, y se nota que por una vez los programadores se pusieron al servicio de los artistas y no al revés. A pesar de que fue muy aplaudida durante la proyección, consiguió el 8º puesto. El muy honorable 6º puesto lo consiguió el grupo español Miracle con su primera producción: Higher Desire. Su demo está dotada de mucha fuerza, mucho ritmo, en resumen, mucha "caña", acompañada por una banda sonora techno con la que todas las imágenes van perfectamente sincronizadas. Si bien no lucía los famosos destellos de Phong, los coders han prometido que su próxima demo estará cargada de Phong. El 7º puesto fue para el grupo Orange, que con la demo Television continuó con su línea de no complex vectors, es decir, efectos 2D con multitud de colores moviéndose por pantalla, túneles y todo tipo de distorsiones, con una estética muy psicodélica. El grupo Capacala, en cambio, presentó Zweilight Zone, una demo puramente 3D, y consiguió el 4º puesto.

En lo referente a las intros, el primer puesto se lo llevó el grupo Wild Light con "Drift", una producción con una estética muy cuidada y con unos efectos de luces y sombras muy conseguidos. El segundo puesto fue para Coma con "Stickman's World". En esta intro puede verse a Stickman, un muñeco hecho con líneas rectas, que enciende la televisión, baila al son de la música y finalmente va a la nevera a por un refresco. Si bien comparada con las restantes intros no tiene ningún mérito técnico, el muñeco se mueve con mucha gracia y la intro es realmente original, algo que hasta ahora estaba escaseando en la escena de demos de PC.

En el concurso de música multicanal consiguió el primer puesto el módulo Catch that Goblin, del célebre Skaven (músico de Future Crew). Una melodía con ambiente de cine y dibujos animados, y de una gran perfección técnica. En la nueva categoría de animaciones, obtuvo el primer puesto "Flow", animación con una calidad comparable a la de los concursos profesionales. El segundo lugar fue para "Pulp" de RRRR & Bang, animación en la que la sencillez y la originalidad triunfan de nuevo sobre la complejidad y la monotonía.

Por otra parte, el curioso concurso de demos de Commodore 64 volvió a sorprender como cada año, al presentarse efectos como plasmas o vectores realizados con un ordenador que tan sólo tiene un procesador de 8 bits y a 1 MHz. Las demos de Amiga, en cambio, que cada año superaban en estética, originalidad y diseño a las de PC, este año sufrieron del mismo defecto de muchas demos de PC: eran todas muy parecidas y predecibles. Quizás esto sea así porque el Assembly es sobre todo un concurso de PCs.

Este año se ha introducido la nueva categoría de animaciones

PRÓXIMAMENTE: LA EUSKAL PARTY III

En resumen, el Assembly ya es una party consagrada, y como consecuencia de ello ya no es una simple reunión de aficionados, sino un acontecimiento de mayor envergadura y con intereses comerciales. Pero precisamente por eso la organización prácticamente no falló en nada. Se cumplió con el horario de proyecciones previsto casi a rajatabla, se pudo disponer de un lugar adecuado para dormir, si una demo fallaba se volvía a proyectar hasta 3 veces, la entrega de premios se hizo en presencia del público, y los premios eran entregados inmediatamente. Es decir, todo lo contrario al Assemlby 94.

Pero tras una party en uno de los lugares más alejados de Europa, se acerca otra en España mismo: la Euskal Party III. Esta party española de Amiga, en su tercera edición, por primera vez tendrá categorías de PC en el concurso. La party tendrá lugar en Tolosa en el puente del Pilar.

PROGRAMACIÓN DE LA TARJETA SOUND BLASTER

Héctor Martínez

Hemos estado tratando siempre con sonidos digitalizados, desde la primera entrega en la que se suministró el primer programa que podía reproducir digitalizaciones, hasta el número anterior en el que llegábamos a interpretar ficheros de formato MOD. Todo esto no es posible si antes no disponemos de los instrumentos y/o voces digitalizadas. Por lo tanto, abriendo un paréntesis en el tema del reproductor de música, pasaremos a ver como podemos nosotros mismos programar la Sound Blaster para digitalizar.

Para poder seguir este artículo con comodidad, es recomendable tener cerca el número 5 de esta revista, ya que se explicó la detección de la tarjeta, la programación del KDMA y la técnica del doble buffer, en el presente artículo se voloverá a hacer referencia a estos puntos.

MODALIDADES DE DIGITALIZACIÓN DE LAS TARJETAS SOUND BLASTER

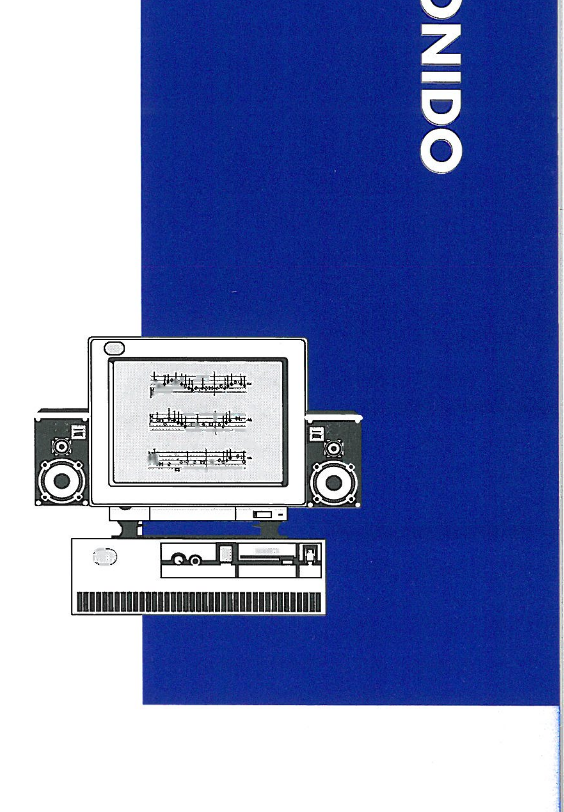
La primera tarjeta Sound Blaster sólo podía digitalizar muestras de 8 bits, en mono y con una frecuencia máxima de 13 KHz. Las generaciones actuales de tarjetas de sonido, han llegado a la calidad de compact disc, es decir, muestras de 16 bits, en estéreo y a 44 KHz. Si queremos sacarle el máximo rendimiento a nuestra tarjeta, habrá que programarla convenientemente para aprovechar todas sus prestaciones. La tabla 1 contiene las posibilidades de digitalización de cada una de las tarjetas de la serie Sound Blaster.

Un aspecto muy importante a la hora de realizar una digitalización es la coherencia entre el modo utilizado y la señal que estamos sometiendo a este proceso. La importancia de esto radica en que ocupa el doble de espacio una muestra digitalizada en estéreo que en mono; al igual ocurre con la frecuencia de muestreo y el número de bits. Por lo tanto, si queremos digitalizar una voz con un micrófono, aunque nuestra tarjeta de sonido soporte 16 bits, estéreo y 44 KHz, lo más conveniente sería digitalizarla en mono, 8 bits y 11 KHz, ya que la entrada de micro de la tarjeta es mono, los 16 bits en una digitalización de una voz no son demasiado importantes y el espectro frecuencial de la voz humana contiene muchas frecuencias en el rango 100 - 4000Hz, y las frecuencias por encima de éstas sólo son los armónicos que varían según el timbre de la persona. Si por el contrario, queremos digitalizar un platillo de una batería, sería importante digitalizar a 44 KHz, ya que este sonido tiene frecuencias que llegan hasta 22 KHz. Hay que tener en cuenta que siempre hay que digitalizar al doble de la frecuencia máxima que queremos abarcar.

ADQUISICIÓN DE SONIDO MEDIANTE DMA

Al igual que para generar sonido, para liberar al procesador de efectuar la transferencia byte a byte de la digitalización, utilizaremos un chip especializado para esta tarea: el DMA (Direct Memory Access).

En el número 5 de esta revista, se explicó minuciosamente como programar el KDMA para transferir datos a un periférico; sólo cabe remarcar, que ahora es el periférico el que tiene que enviar muestras hacia la CPU, y por lo tanto hemos de indicarle al KDMA que el sentido de transferencia será éste.



Cualquier tarjeta de sonido que se precie dispone de la capacidad de digitalizar señales externas. Esta señal digitalizada se puede guardar para posteriormente generar ese sonido cada vez que interese, e incluso para aplicar un tratamiento digital y añadirle eco, cambiar las frecuencias, invertirlas, o mediante técnicas de inteligencia artificial llegar al reconocimiento de voz.

Nombre	Versión DSP	Bits por muestra	Stereo	Frecuencia Máxima	Mixer	Modo auto-inicializado
Sound Blaster	1.xx	8	NO	13 KHz	NO	NO
	2.00	8	NO	13 KHz	NO	SI
Sound Blaster 2.0	2.01+	8	NO	15 KHz	NO	SI
Sound Blaster Pro	3.xx	8	NO	44 KHz	SI	SI
Sound Blaster Pro	3.xx	8	SI	22 KHz	SI	SI
Sound Blaster 16	4.xx	8/16	SI	44 KHz	SI	SI

Tabla 1: Características de digitalización segun el tipo de Sound Blaster.

Esto se hace mediante el registro de modo (puerto 0Bh para el DMA de 8 bits y D6h para el de 16). Los bits 2 y 3 indicaban el sentido de la transferencia: 01 para una transferencia de periférico a memoria, y 10 para una transferencia de memoria a periférico. Por lo tanto para programar el KDMA utilizaremos el procedimiento *ProgKDMA* pasándole como parámetro *DesDeMem* el valor False.

La pauta a seguir para programar la Sound Blaster en modo ADC es:

1. Programación del KDMA para transferir N*2 bytes desde la tarjeta a la dirección de memoria D (en donde se supone que estará el buffer reservado para la digitalización), por el canal de DMA asignado a la SB.

2. Programación del DSP para enviar N bytes por DMA. A partir de este momento la SB comenzará a digitalizar.

3. Al finalizar la transferencia, la SB produce una interrupción del nivel de IRQ que tenga asignado. La rutina de servicio a la interrupción debe confirmar, tanto al controlador de interrupciones como a la SB, que la interrupción ha sido atendida. La confirmación a la SB se realiza efectuando una lectura del puerto Base + Eh si estamos digitalizando a 8 bits, y del puerto Base + Fh si lo estamos haciendo a 16 bits. Por supuesto, siguen existiendo las mismas restricciones que habían en el modo DAC, es decir, que no podremos transferir bloques mas grandes de 64 K, y que el buffer que utilizará el DMA no puede cruzar una página física del KDMA. Para poder estar digitalizando indefinidamente, volveremos a utilizar la técnica del doble buffer ya explicada en el número 5 de la revista. De esta manera, con un buffer relativamente pequeño podremos adquirir tantos datos como nos sea preciso. Si nos interesara guardar estos datos en otra zona de memoria, sólo habría que copiar el último buffer llenado en la nueva zona de memoria. Al igual que ocurría al enviarle datos al DAC, el programa principal puede ir haciendo otras tareas paralelamente.

PROGRAMACIÓN DEL DSP PARA LA TRANSFERENCIA EN MODO ADC

Una vez listo el KDMA para la transferencia, hay que programar la SB para que envíe los bytes por DMA y los suministre a la CPU. En primer lugar, al contrario que en modo DAC, hay que deshabilitar la salida de sonido digitalizado escribiendo el comando D3h en el DSP. En la Sound Blaster 16 no hace falta este comando, de hecho lo ignora, por lo que si se lo enviamos tampoco pasa nada. También hay que programar la constante de tiempo, a través de la cual la SB determinará la frecuencia de muestreo. La fórmula para calcular la constante de tiempo de la SB es la misma que se utilizaba para generar sonido, de manera que la rutina que tenemos para calcularla no necesita ninguna modificación:

Constante de tiempo = 256 - (1.000.000 / Frecuencia de muestreo)

Se programa escribiendo en el DSP el comando 40h seguido de un byte con el valor de la constante.

Hay que tener en cuenta que para las tarjetas Sound Blaster Pro, si se programan en modo estéreo, la frecuencia de muestreo que hemos de utilizar en esta fórmula es el doble que la deseada, como ya se comentó en el número anterior.

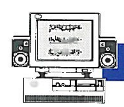
Con los DSP de versiones 3.xx, podemos digitalizar en estéreo a una frecuencia de hasta 22 KHz, para ello, hay que enviar al DSP el comando A8h para poner el modo ADC en estéreo. Las versiones de DSP 4.xx ignoran este comando, ya que se programan de forma totalmente diferente.

Luego hay que programar el tamaño del buffer de la misma manera que se hacía en modo DAC, es decir, hay que especificar el número de bytes que tienen que transferirse para que la tarjeta genere una interrupción, que en este caso será la mitad del tamaño del buffer de DMA. En este punto, cada tipo de SB se programa de forma distinta. En el caso de los DSP de versión entre 2.00 y 3.xx, se utiliza el comando 48h seguido del tamaño del buffer menos uno (primero el byte bajo y luego el byte alto). En los DSP de versión 1.xx este comando no existe y no se ha de utilizar, ya que se envía el tamaño del buffer junto con el comando que inicia la transferencia. En la versión 4.xx, el comando existe por compatibilidad, pero no es conveniente programarlo así, y es mucho mejor hacerlo como se verá posteriormente

La primera tarjeta Sound Blaster sólo podía digitalizar muestras de ocho bits

Finalmente hay que indicar a la tarjeta que inicie la transferencia. También en este caso se utiliza un comando distinto para cada tipo de SB:

- Los DSP de versión 1.xx se programan enviando el comando 24h seguido del byte bajo y el byte alto del tamaño del buffer menos uno. Además, como estas tarjetas no dispo-



nen del modo auto-inicializado, tras cada interrupción hay que volver a enviar el mismo comando y los mismos datos al DSP para que se reanude la transferencia.

- Con el DSP de versión 2.00, una vez programado el tamaño del buffer con el comando 48h, se inicia la transferencia enviando el comando 2Ch.

- Con los DSP de versiones comprendidas entre la 2.01 y 3.xx, una vez programado el tamaño del buffer, se inicia la transferencia enviando el comando 98h. Este comando pone el DSP en modo de alta velocidad, de forma que se pueden conseguir frecuencias de muestreo mayores que con versiones anteriores del DSP.

- El DSP de la SB 16 (versión 4.xx), se programa enviando el comando CEh para 8 bits o BEh para 16 bits, seguido de un byte que indica la modalidad: mono/stereo con signo/sin signo (ver tabla 2), y seguido de 2 bytes que indican el tamaño del buffer menos uno.

RUTINA DE TRANSFERENCIA

La función *SBRecBuf* se encarga de todo lo explicado, realizando las siguientes tareas:

1. Calcula la constante de tiempo en función de la frecuencia de muestreo y programa al DSP con la constante calculada.

2. Ubica una rutina de servicio a interrupción en la interrupción correspondiente a la SB, y programa al controlador de interrupciones para habilitarla. La ISR confirma al controlador de interrupciones y a la SB que la interrupción ha sido atendida, y llama a una rutina especificada por el usuario que se encargará de actualizar el buffer.

3. Programa el controlador de DMA para transferir 2*N bytes.

4. Programa la SB para que genere una interrupción cada N bytes transferidos.

5. Inicializa la transferencia.

Los pasos 4 y 5 varían según la versión del DSP, pero básicamente ésta es la estructura de la rutina.

Modo	Byte
mono sin signo	00h
estéreo sin signo	20h
mono con signo	10h
estéreo con signo	30h

Tabla 2: Byte que indica la modalidad para mono/estéreo y con signo/sin signo.

La Sound Blaster 16 permite muestrear y reproducir al mismo tiempo

PECULIARIDAD DE LA SOUND BLASTER 16 BITS

Esta tarjeta, al disponer de dos canales de DMA (uno de 8 bits y otro de 16) nos permitirá programar un canal de memoria a periférico y el otro de periférico a memoria. De esta manera se puede digitalizar una señal, reconvertirla, y en tiempo real, volver a generar el sonido tratado, con lo que se podrían hacer ecos y otros efectos sonoros. La manera de hacerlo sería:

1. Crear dos buffers, uno de entrada y otro de salida. Uno de ellos ha de ser el doble de grande que el otro, ya que habrá que digitalizar a 8 bits y generar sonido a 16 bits o viceversa.

2. Llamar al procedimiento *SBRecBuf* pasándole como parámetro el buffer de entrada y habiendo puesto la variable *Sb16BitsRec* a False.

3. Llamar al procedimiento *SBPlayBuf* pasándole como parámetro el buffer de salida y actualizando la variable *Sb16Bits* a True. (La frecuencia de muestreo a la que se reproduce y se muestrea tiene que ser la misma).

4. En la rutina de servicio a la interrupción hay que realizar el tratamiento de la señal, y una vez tratada, irla copiando en el buffer de salida.

El problema que todo esto puede traer consigo, es que programamos la Sound Blaster dos veces y sólo tenemos una interrupción disponible para la tarjeta. Esto quiere decir que se ejecutará la rutina de servicio a la interrupción tanto en el momento en que se llene un buffer al digitalizar, como cuando se haya enviado un buffer muestreando, de manera que en lugar de llegar una interrupción nos llegarían dos y además el *acknowledge* que hay que hacerle al DSP es diferente según si estamos utilizando 8 o 16 bits.

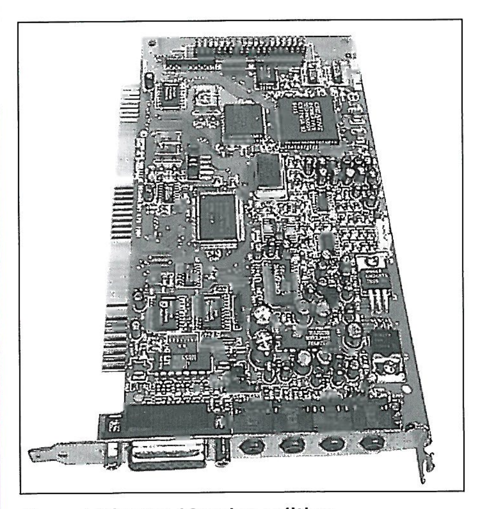
La solución a todo esto está en consultar el registro 82h del mixer de la tarjeta. Este registro tiene activado un bit dependiendo del tipo de interrupción que haya ocurrido: si está activado el bit 0, indica que la causa ha sido una transferencia de muestras de 8 bits . Por el contrario, si el bit activo es el 1, indica que la transferencia era de 16 bits.

Para leer de este registro, primero hay que escribir en la dirección Base + 4h el valor 82h, y posteriormente leer de la dirección Base + 5h (ya se extenderá esta información en números posteriores cuando se hable del mixer de las tarjetas Sound Blaster Pro y 16 bits).

Por lo tanto habrá que modificar ligeramente la rutina de atención a la interrupción que había en las otras entregas para que compruebe este registro y actúe según convenga. La manera de hacerlo es la siguiente, si suponemos que estamos haciendo una transferencia utilizando el DMA de 16 bits:

IF (SBReadMixer($82) AND 2)=2 THEN {Ha llegado una interrupción causada}
{Por una transferencia a 16 bits}

- Hacer el tratamiento del buffer de 16 bits.
- Hacer el Ack al DSP



Sound Blaster 16 value edition.

- Enviar EOI al Controlador de Interrupciones

ELSE

- Llamar a la rutina de interrupción que había antes que ésta

ENDIF

EL PROGRAMA DE EJEMPLO

El programa de ejemplo ADC.PAS suministrado con este artículo, se encarga de digitalizar una señal y mostrarla en un osciloscopio. Dibujar el osciloscopio de una señal es muy sencillo: todo consiste en ir incrementando la coordenada X, y como coordenada Y ir tomando sucesivamente un valor tras otro de la muestra. Como la pantalla no es infinita y tiene una cierta resolución, cada vez que llegamos al extremo derecho de la pantalla hay que volver a poner la X a 0 y esperar a que se llene un nuevo buffer para volver a dibujar.

Si lo hacemos así, veríamos como la onda no se está quieta, y parece que se desplace o incluso parpadee. Esto se debe a que no empezamos a dibujar esta siempre desde el mismo punto. Si queremos conseguir un osciloscopio más real lo que debemos hacer es sincronizarnos con la onda y empezar a dibujar cuando la onda pasa siempre por el mismo sitio. Un punto a tomar como referencia puede ser el 0. Por ejemplo podemos esperar a que la onda pase de negativo a positivo para empezar a dibujar. Esto es lo que hace la función synch del programa de ejemplo. Es interesante probar la ejecución del programa sin ejecutar el procedimiento synch para ver el efecto que produce.

Para los usuarios que dispongan de una Sound Blaster 16, se ha implementado una pequeña demostración de lo que se puede llegar a hacer programando la tarjeta en los modos ADC y DAC simultáneamente.

El programa ejemplo copia el buffer de entrada en el buffer de salida, pero no lo copia a la misma frecuencia, sino que lo hace saltando o repitiendo muestras, al igual que lo hacíamos con cada instrumento cuando interpretábamos un fichero MOD. El efecto que se consigue es cambiar la voz del que habla por el micrófono, de manera que se oirá una voz de "monstruo" o de "pitufo" según si hemos bajado o subido la frecuencia..

Para cambiar el tono de la voz, se utilizan las teclas '+' y '-' para subir y bajar la frecuencia respectivamente, y la tecla de espacio para restaurar la frecuencia inicial.

El programa de ejemplo permite alterar el tono de voz en tiempo real

Como los datos muestreados por el programa son enviados hacia el mixer de la tarjeta por el canal voice, y a su vez, la señal original por el canal Mic, lo que oiremos a través de los altavoces serán la señal original más la señal reconvertida. Para evitar esto, ejecutar el programa Sb16Set suministrado con el software de la tarjeta, y desactivar la salida del micrófono. Esto mismo lo debería hacer el programa ejemplo automáticamente, pero como todavía no se ha explicado el funcionamiento del mixer, no se ha incluido.

Si ejecutamos el programa ADC y pulsamos el '+', notaremos como la voz sale mas aguda, pero no se oye como cabía esperar (se escucha una reverberación desagradable). Esto es debido a que si el buffer de entrada lo copiamos en el buffer de salida saltándonos alguna muestra de vez en cuando, se provocan cortes bruscos en la onda original. Además de esto, llegaremos al final de la muestra de entrada antes de que hallamos llenado el buffer de salida; y para acabar de rellenar lo que falta, se vuelve a empezar desde el comienzo del buffer de entrada. Por lo tanto, estamos enviando las muestras más de una vez. Si por el contrario pulsamos la tecla '-', el buffer se recorre repitiendo muestras y nunca se llegará al final del buffer de entrada, y esto provoca que dejemos siempre algo de lo que hemos digitalizado sin muestrear. La manera ideal de conseguir estos efectos con una calidad aceptable, es utilizando técnicas de tratamiento digital de la señal más avanzada, pero como programa de ejemplo éste es bastante sencillo y educativo, y consume mucho menos procesador.

EN LA SIGUIENTE ENTREGA

En este artículo se han explicado algunas cuestiones que quedaban pendientes sobre el funcionamiento de las distintas variantes de la tarjeta Sound Blaster, pero todavía nos queda la programación del mixer de las tarjetas Sound Blaster Pro y 16.

También hay que completar más nuestro reproductor. Faltan por implementar algunos efectos, un control de saturación, volúmenes independientes para cada canal, se podría añadir una interpolación por software para acercarnos a la calidad de la GUS y soportar ficheros en formato S3M. Posiblemente añadiendo todo esto a nuestro player comience a ir lento el programa en primer plano que se esté ejecutando y haga falta optimizar las rutinas de mezcla.



IDL 3.6.1

Fernando de la Villa

En realidad IDL no es un lenguaje de programación propiamente dicho. Más bien se puede definir como un completo sistema para el análisis y visualización de datos de forma interactiva. Incluye un lenguaje de programación de cuarta generación similar al lenguaje APL, y un conjunto de elementos (widgets) para la creación de interfaces gráficos de usuario(GUI).

Además, dispone de numerosas funciones orientadas al análisis matemático, proceso de imágenes y técnicas de visualización, lo que le convierte en una poderosa herramienta de tratamiento de imágenes.

El lenguaje de datos interactivo ofrece una alternativa a la programación en FORTRAN o C, reduciendo considerablemente el tiempo de desarrollo de una aplicación de tipo científico. Es, al mismo tiempo, una herramienta de estudio. Su naturaleza interactiva permite al usuario trabajar con los datos y obtener resultados de forma inmediata, posibilitando la realización de los cambios y ajustes necesarios antes de proseguir con el trabajo.

El paquete está disponible en las siguientes plataformas de desarrollo: Windows, Windows NT, Macintosh, Power Macintosh y existen versiones UNIX y VMS. Una aplicación escrita con IDL en cualquiera de estas plataformas puede ser transportada a otra sin apenas realizar cambios.

EL LENGUAJE

IDL incorpora un lenguaje de programación orientado al array, es decir, que los operadores y funciones trabajan directamente sobre conjuntos de datos. Incluye funciones, procedimientos y estructuras de control, y contempla la gestión de errores y la creación de librerías. También permite el uso de funciones externas escritas con otros lenguajes de programación.

Los tipos de datos del lenguaje están orientados al manejo de datos de tipo numérico. Soporta bytes, enteros, enteros largos, números en coma flotante, complejos, cadenas y estructuras, así como una combinación de los anteriores. Las variables en IDL son dinámicas, por lo que su estructura y tipo de dato pueden ser cambiados durante la ejecución.

La sintaxis de las estructuras de control es muy similar a las de C y FORTRAN. Dispone de las estructuras repetitivas for, while y repeat, las alternativas if...then e if...then...else y la selectiva case. Admite la creación de sentencias complejas con el uso del bloque begin...end. En total, IDL tiene doce tipos de sentencias diferentes.

IDL dispone de numerosas funciones orientadas al análisis matemático, proceso de imágenes y técnicas de visualización

El desarrollo de una aplicación de tipo científico puede llegar a ser muy complicado debido al intensivo tratamiento de datos e imágenes necesario. Lo mejor en estos casos es utilizar una herramienta de desarrollo especializada como IDL, el lenguaje de datos interactivo.

FUNCIONES Y PROCEDIMIENTOS

Una de las características más completas de IDL es su librería de funciones y procedimientos. Ofrece un gran número de funciones de diversa índole, clasificadas en diferentes grupos:

- Animación

Con IDL se pueden realizar animaciones con las imágenes obtenidas de los cálculos. Esto es especialmente útil cuando el conjunto de datos en estudio corresponde a un espacio tridimensional. Las funciones Xanimate y Xinteranimate son las encargadas de mostrar una secuencia animada de imágenes previamente creadas.

- Arrays

Como ya se dijo anteriormente el lenguaje está orientado al manejo de conjuntos de datos, razón por la que las funciones orientadas al tratamiento de arrays cobran una especial importancia. Existen funciones para buscar "regiones" de datos similares en una imagen (matriz bidimensional), cálculo de la función densidad, búsqueda de máximos y mínimos, rotaciones, desplazamientos y un largo etcétera.

- Ficheros

En un lenguaje de manejo de datos no podían faltar las funciones de tratamiento de ficheros. IDL dispone de todas las operaciones básicas necesarias de apertura, cierre y lectura/escritura, además de otras más especializadas como una función de búsqueda de ficheros.

- Fuentes de letras

Para la realización de anotaciones sobre las imágenes y la salida de texto en pantalla se ofrece un método independiente del hardware: fuentes escalables. Son tipos de letras dibujados mediante líneas cuyo aspecto en pantalla será el mismo sin importar la plataforma en que se esté ejecutando IDL. Existen cinco funciones para la gestión de fuentes de letras y varios comandos de tipo cadena para la impresión en pantalla de los textos.

- Gráficos

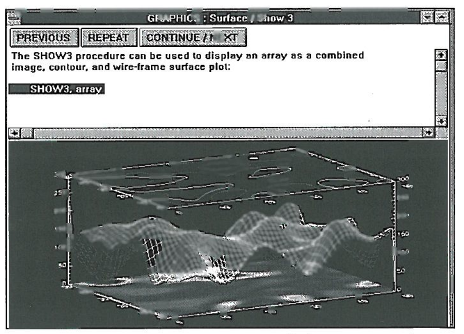
Aparte de las funciones más especializadas, IDL posee funciones gráficas de tipo genérico. Algunas de las funciones de este grupo permiten el dibujo de flechas, la definición de Regiones De Interés (RDI), el escalado tridimensional, y el acceso a funciones específicas del dispositivo gráfico. También hay funciones para la manipulación de la paleta gráfica, como la corrección gamma, cuantización, reducción y conversión de colores.

- Visualización de imágenes

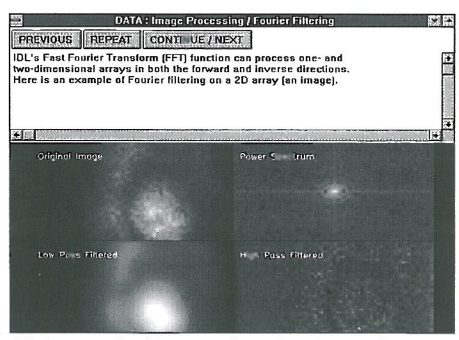
Uno de los aspectos más importantes del estudio de un conjunto de datos es la representación gráfica de los resultados. El lenguaje de datos interactivo proporciona funciones para la visualización, escalado, ampliación y reducción de imágenes.

- Proceso de imágenes

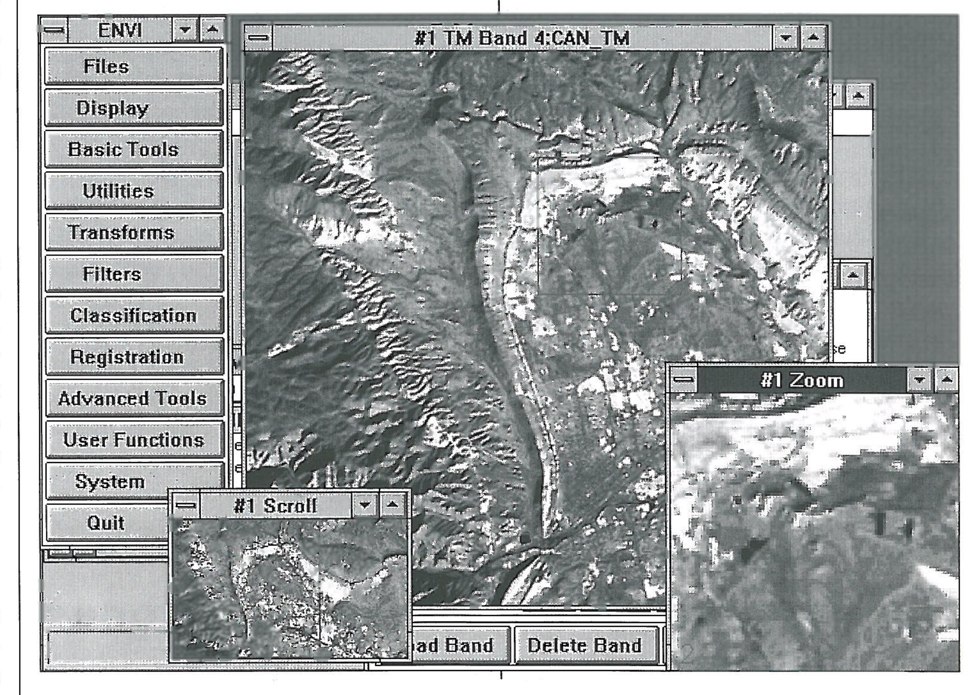
Para la obtención de información interesante acerca de la imagen resultante



Junto al programa se suministra una interesante demostración escrita con el propio IDL que muestra las capacidades del lenguaje.

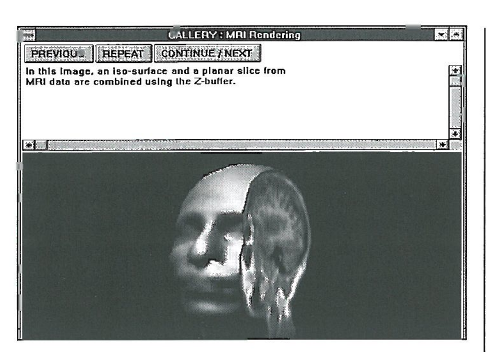


El lenguaje interactivo de datos incluye capacidades de filtrado de imágenes y de paso de imágenes al dominio frecuencial.



ENVI es un avanzado sistema de procesamiento de imagen para el estudio de datos de teledetección creado íntegramente con IDL.

El paquete está disponible en varias plataformas de desarrollo diferentes como Windows, Macintosh y UNIX



La medicina es sólo uno de los posibles campos de aplicación de este lenguaje orientado al análisis y visualización de datos.

de un cálculo, o de una fotografía digitalizada suele ser necesario algún tipo de procesamiento, especialmente en el caso de las imágenes a color. La librería de funciones contiene métodos avanzados para el proceso de imágenes como filtros digitales no recursivos, dilación y erosión morfológica, transformada rápida de Fourier, perfilado de bordes y suavizado de la imagen, entre otros.

- Entrada/Salida

Una herramienta de las características de IDL no estaría completa sin un conjunto de funciones de manejo de diferentes formatos gráficos. Aparte de las funciones de entrada de datos desde el teclado y salida en pantalla, IDL incorpora funciones para leer y escribir imágenes en formato TIFF, GIF, PICT, BMP y JPEG, además de multitud de funciones específicas de los formatos CDF, netCDF y HDF.

- Mapas

Otra de las áreas claramente diferenciadas del lenguaje de datos interactivo es su capacidad de manejo de mapas. Está orientada al estudio del globo terráqueo, y permite cálculo de distancias, diversos tipos de proyecciones y trazado de paralelos y meridianos.

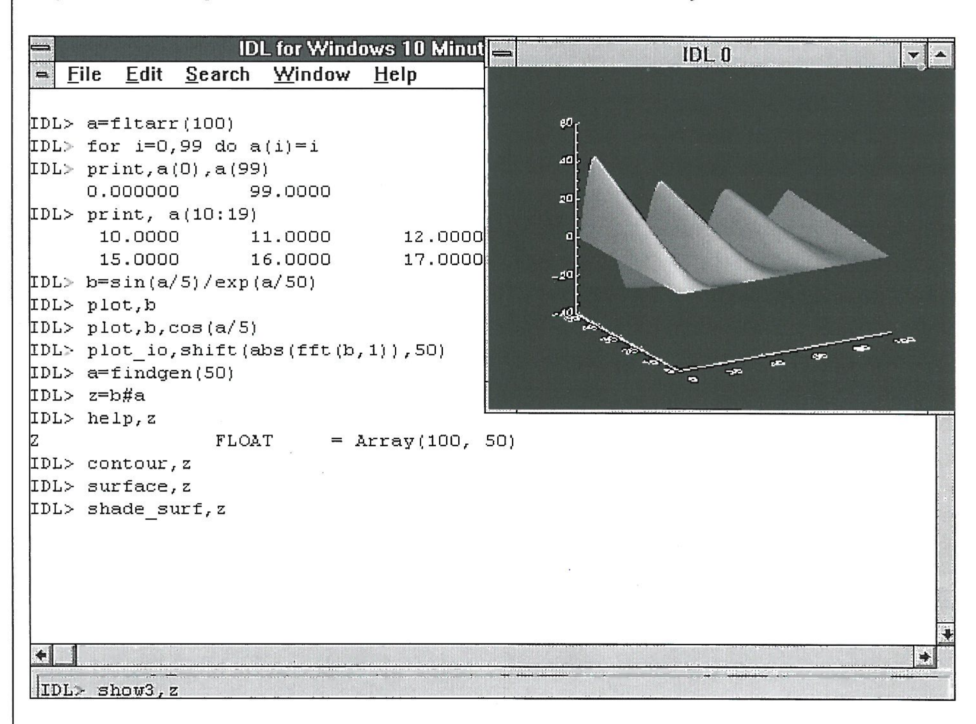
Una de las características más completas de IDL es su librería de funciones y procedimientos

- Matemáticas

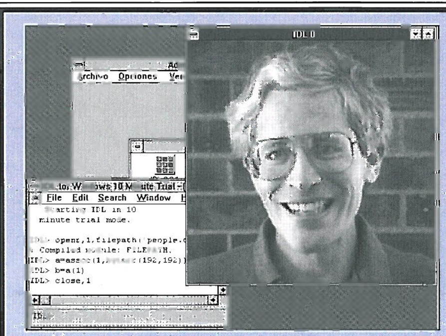
Entre los puntos fuertes de IDL se encuentra su gran capacidad de cálculo. Hay funciones matemáticas básicas, especiales, trascendentales, genéricas y de "recetas numéricas". Por citar algunas de las posibilidades, con estas funciones se puede calcular el sumatorio de Riemann, resolver un sistema de ecuaciones lineales por el método de Cramer, resolver ecuaciones diferenciales por Runge-Kutta, obtener la integral de una función sobre un intervalo abierto o cerrado e interpolar valores en un vector.

- Estadística

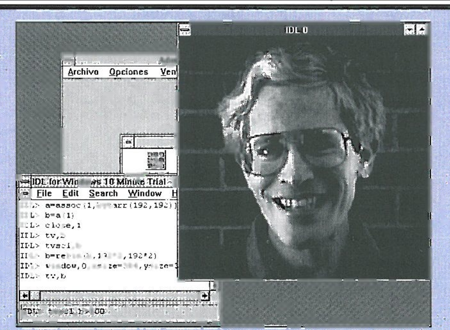
Junto a las funciones de cálculo matemático se incluyen diversas funciones



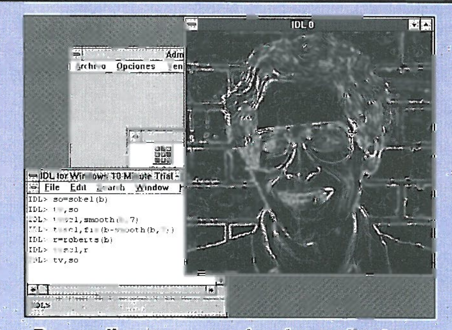
El usuario puede crear un conjunto de datos y mostrar gráficamente los resultados de forma interactiva en muy pocos minutos.



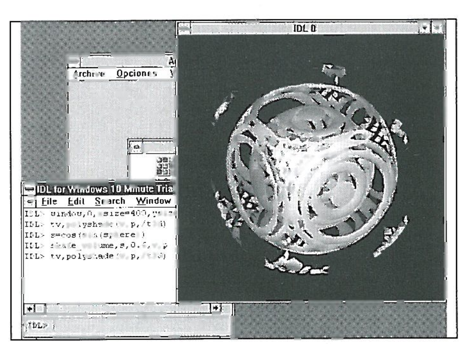
IDL trata las imágenes como arrays bidimensionales de datos sobre los que se pueden realizar distintos tratamientos.



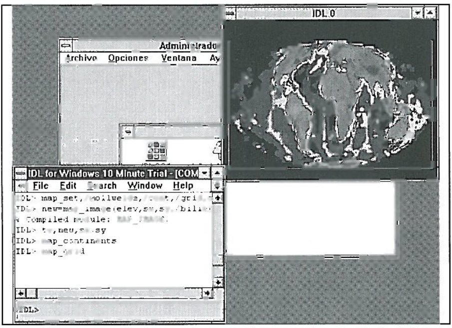
Mediante un sencillo comando de visualización se puede mostrar la imágen anterior con el contraste variado.



Para aplicar un operador de gradiente sobre la imágen basta con utilizar la función adecuada sobre la matriz bidimensional.



IDL también puede ser utilizado para el estudio de volúmenes tridimensionales tan curiosos como el de la imagen.



Uno de los aspectos más curiosos del lenguaje es la inclusión de funciones específicas para tratamiento de mapas.

FICHA

Nombre:	IDL 3.6.1 for Windows
Fabricante:	Research Systems Inc.

Requerimientos:
- Windows 3.1 o superior
- Procesador 386 o superior

Ratón

Distribuidor:	Estudio Atlas, S. L.
Precio:	255.000 pts

estadísticas, para un adecuado estudio de datos. Permite la obtención de la distribución binomial, distribución de Student, coeficiente de correlación, regresión, desviación típica, etc.

Aparte de estos grupos de funciones existen otros como conversión de tipos de variable, tratamiento de cadenas y acceso al sistema operativo.

GESTIÓN DE ERRORES

IDL sigue la tendencia de los lenguajes modernos de incorporar un sistema de manejo errores. Los errores los tiene divididos en tres categorías:

- Errores de entrada/salida
- Errores de cálculo
- Otros (errores de programación o errores producidos en tiempo de ejecución)

La captura de errores se realiza mediante la inclusión de las cláusulas On_IOError para los errores de entrada/salida, y On_Error o Catch para los errores genéricos. Catch ha sido introducido en la versión 3.6 y constituye un método más robusto de manejo de errores que On_IOError y On_Error.

Proporciona una manera sencilla de creación de interfaces gráficos de usuario y de gestión de eventos

Cuando se produce un error en un módulo controlado por Catch, IDL actualiza la variable del sistema que contiene el valor numérico del error producido, y busca un gestor de errores (Catch). Si no lo encuentra, el programa es interrumpido con un mensaje de error. Si lo encuentra, la ejecución continúa en la línea siguiente al catch, que deberá ser la porción de código correspondiente al código de respuesta ante el error.

CONSTRUCCIÓN DE INTERFACES GRÁFICOS DE USUARIO

Sin duda alguna, la tarea más pesada y repetitiva para los programadores es la creación de los interfaces (IGU, Interfaz Gráfico de Usuario). Para ayudar en esta tarea, IDL proporciona una manera sencilla de creación de interfaces, y gestión de eventos. El lenguaje soporta siete tipos de widgets o elementos: base, botón, deslizador, texto, dibujo, etiqueta y lista. Los pasos para crear un IGU para manejo de un programa son los siguientes:

Incorpora funciones para leer y escribir imágenes en formato TIFF, GIF, PICT, BMP y JPEG

1.- Creación de la jerarquía de widgets
2.- Visualización de los widgets
3.- Preparación del programa para responder a eventos
4.- Realización del control de widgets
5.- Obtención de información sobre los widgets
6.- Destrucción de los widgets

Todo lo anterior se traduce en un número pequeño de líneas de código gracias a las funciones de librería incluidas en IDL para el manejo de widgets.

LLAMADAS A PROCEDIMIENTO DE FORMA REMOTA

La última característica destacable del lenguaje de datos interactivo es su capacidad de actuar como servidor RPC, posibilitando que otras aplicaciones se comuniquen con él. La implementación del mecanismo RPC consiste en que IDL actúa como servidor RPC y un programa de usuario funciona como cliente. El programa cliente realiza peticiones al servidor, ejecutando las sentencias dentro del propio IDL.

EN DEFINITIVA

El lenguaje de datos interactivo es una herramienta especializada, y a la vez muy completa y fácil de usar. Resulta adecuada para usuarios avanzados que no están acostumbrados a programar de forma intensiva, y constituye una excelente herramienta de estudio para ingenieros aprovechando las capacidades gráficas y de cálculo de un ordenador.

CURSO DE RAY TRACING (I): INTRODUCCIÓN

Alvaro Silgado

¿Quién no ha visto alguna vez en televisión esas cabeceras de programas o imágenes increíblemente perfectas de escenarios y objetos inexistentes? Las técnicas gráficas están de moda y se emplean cada vez más. Incluso se está llegando a sustituir platós de televisión por mundos virtuales generados en tiempo real por potentes ordenadores. Y eso no es nada comparado con lo que se puede ver en el cine: hombres que se transforman en seres de mercurio, lenguas de agua que permanecen en suspensión, dinosaurios, etc.

Todos estos efectos y todos los que se le puedan ocurrir pueden ser creados mediante técnicas gráficas como el ray tracing. Con el ray tracing se pueden diseñar escenas de todo tipo, y obtener imágenes de ellas con un realismo sorprendente.

A lo largo de este curso se van a estudiar todos los aspectos de esta técnica de síntesis de imagen, y al mismo tiempo, se irá construyendo un programa que cada vez generará imágenes más reales hasta obtener una calidad verdaderamente profesional. Sólo será necesario un ordenador lo más rápido posible, un compilador de C++ y un poco de creatividad. La velocidad del ordenador es un factor a tener en cuenta, ya que el tiempo de generación de una imagen con ray tracing puede oscilar entre unos minutos y varias horas, dependiendo de su complejidad y del tamaño de la imagen a generar.

Para la comprensión del programa, se asume que el lector tiene ciertos conocimientos de programación en C++. Cualquier duda sobre este lenguaje de programación puede ser consultada en el curso de C++ que está siendo publicado en esta revista.

TRAZANDO RAYOS

El objetivo de la técnica es crear una serie de objetos que componen una escena, y poder introducir una cámara virtual que "fotografíe" dicha escena. El "papel fotográfico" en el que quedará plasmada la imagen, será la pantalla del ordenador. Sólo es necesario saber de qué color será cada píxel que compone la imagen.

Como su nombre indica, el ray tracing consiste en trazar rayos. Estos rayos son las trayectorias que siguen los fotones de luz incidiendo en la cámara que capta la imagen. En realidad, la luz parte de una fuente (lámpara, sol, etc.) y se propaga en todas las direcciones. Al incidir sobre un objeto y dependiendo del material del que está hecho, en parte es absorbida, en parte reflejada y en parte refractada. A su vez, los rayos reflejados y refractados inciden contra otros objetos y van rebotando hasta que alguno incide sobre la cámara.

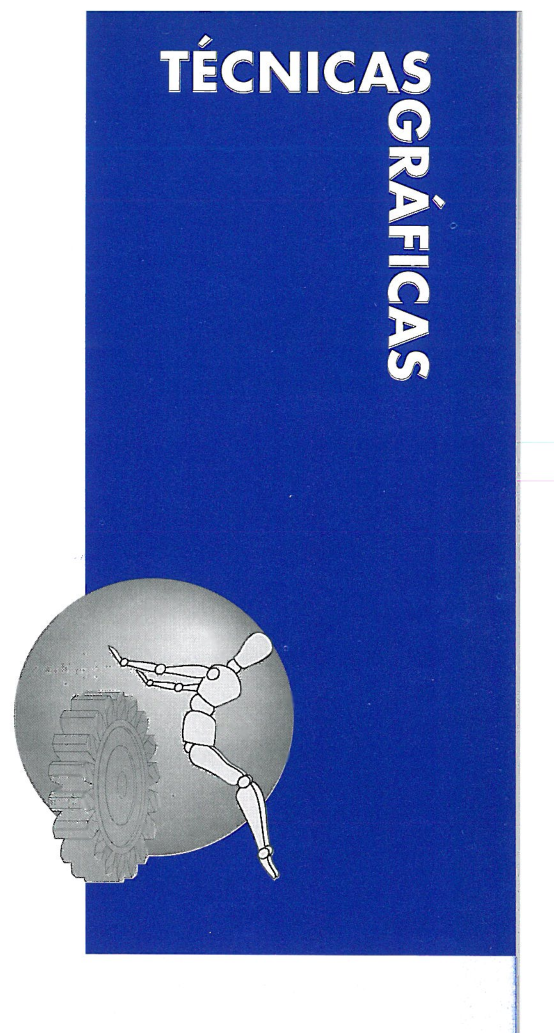
Por lo tanto, para obtener el color de un píxel de la cámara es necesario saber qué rayos de los infinitos emitidos por cada luz terminan pasando por él. Evidentemente, no se pueden estudiar infinitos rayos y seguir sus trayectorias para ver si terminan pasando por el píxel. En lugar de eso, se hace el proceso inverso.

RAY TRACING, PERO AL REVÉS

El proceso de obtener el color de un píxel mediante ray tracing inverso es el siguiente:

- Se traza un rayo que parte del observador y atraviesa la pantalla por el píxel que se está estudiando.

- Se comprueba si ese rayo colisiona con algún objeto de los que componen la escena.



El ray tracing es una técnica mediante la cual se pueden diseñar escenarios fantásticos y obtener imágenes de ellos, con la misma calidad y realidad que en una fotografía. Adéntrese en el mundo de la imagen hiper realista, ha llegado la hora de crear.

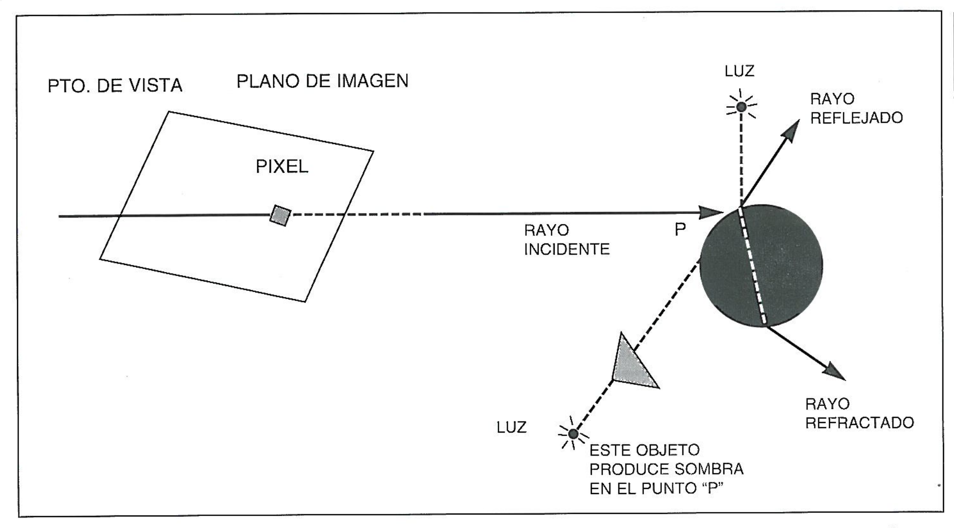


Figura 1. Ray Tracing inverso. Los rayos parten del punto de vista y pasan por el píxel a estudiar.

- Si no colisiona, el color de ese píxel es, por ejemplo, el del cielo.

- Si colisiona, se comprueba si el punto de intersección está siendo iluminado por algún objeto luminoso o lámpara.

- Para ver si un punto está siendo iluminado por una lámpara, se traza un rayo desde el punto hasta ésta y se comprueba si algún otro objeto está situado en su trayectoria. Si es así, el punto está en sombra. Si no, el punto está iluminado.

- Una vez obtenida la luz directa del punto, se le suma una luz ambiental, que se considera constante para todos los elementos de la escena.

- Si el objeto no refleja luz ni la refracta (como la goma, por ejemplo), el color del píxel es el del objeto, más o menos atenuado por la luz.

- Si el objeto refleja la luz (como el plástico pulido), será necesario calcular el rayo reflejado y estudiar qué color viene por él. El color del píxel será el calculado anteriormente más el color del rayo reflejado. Esto se verá con detalle en próximos capítulos.

- Si el objeto refracta la luz (como el cristal), el proceso será similar, pero con el rayo refractado. También se estudiará más adelante. Este proceso queda representado en la figura 1.

A VUELTAS CON LA GEOMETRÍA

Antes de continuar, es necesario establecer una serie de criterios para el diseño de la escena. Como se puede apreciar en la figura 2, hay dos sistemas de coordenadas: uno para el mundo y otro para la cámara. La cámara se encuentra situada dentro del mundo, así que su posición estará descrita según el sistema de coordenadas de éste.

En el mundo, se establece un punto de referencia u origen de coordenadas y tres ejes: X, Y y Z. El eje Y es el que describe la altura del objeto, mientras que X y Z describen su posición en el plano horizontal. En la cámara, el origen de coordenadas está situado en el centro de la pantalla, el eje H (horizontal) es positivo hacia la derecha, el eje V (vertical) es positivo hacia arriba y el eje Z (profundidad) es positivo "entrando por el monitor". O dicho de otra manera, cuanto más alejado está un objeto dentro de la imagen, mayor es su coordenada Z según el sistema de coordenadas de la cámara.

LOS VECTORES, ELEMENTOS BÁSICOS

Un vector es una clase formada por tres variables reales, que pueden servir para representar un punto (su posición), un vector director o un color (sus componentes rojo, verde y azul).

Además de éste, todos los demás elementos que se van a tratar están programados como clases. La ventaja de esto es la claridad del programa y la facilidad de su mantenimiento.

Además de definir la clase, se han escrito operadores para trabajar con facilidad con elementos de este tipo. De esta forma, se pueden escribir expresiones como: a = b + c, en donde a, b y c son de clase vector. También se han programado como operadores el producto escalar (|), y el producto vectorial (&) de dos vectores.

Antes se ha dicho que uno de los usos de esta clase es para definir vectores directores. Como su propio nombre indica, un vector director sirve para indicar una dirección (un eje, una trayectoria, etc.). Hay que destacar que todos los vectores directores que se empleen irán normalizados, es decir, su normal (o longitud) vale exactamente 1. Para normalizar un vector, primero se calcula su normal, que es la raíz cuadrada de x^2 + y^2 + z^2, y luego se divide cada componente por la normal obtenida.

LOS RAYOS, LA CLAVE DE TODO

Un rayo es un vector con la particularidad de tener un punto de partida. Es decir, está formado por dos elementos de la clase vector: uno para indicar su origen y otro para su dirección. Si se denomina al origen como Ro (xo, yo, zo), y a la dirección como Rd (xd, yd, zd), se puede definir un rayo como un conjunto infinito de puntos de la línea

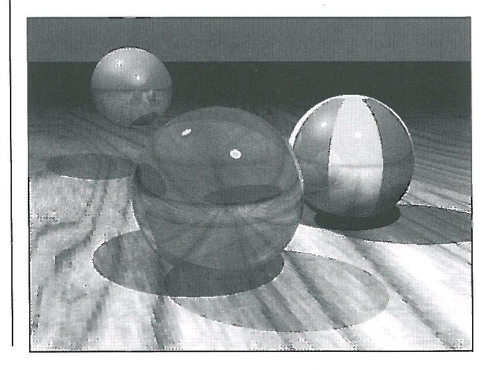
R(t) = Ro + Rd * t, con t > 0.

La ventaja de esta definición es que si un punto P (xp, yp, zp) pertenece a un rayo, es muy sencillo calcular la distancia a la que se encuentra del origen de éste. No hay más que despejar en la fórmula anterior, obteniéndose:

t = (xp - xo) / xd, o
t = (yp - yo) / yd, o
t = (zp - zo) / zd

Como se verá más adelante, esto va a ser muy útil al hacer la intersección de un rayo con un objeto.

Se han programado tres funciones básicas de tratamiento de rayos. La primera permite generar un rayo que parte



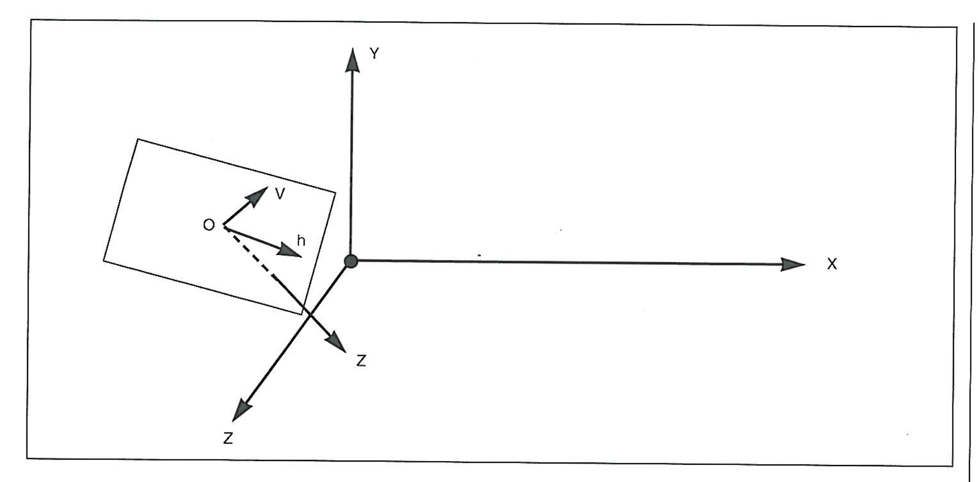


Figura 2. Hay un sistema de coordenadas para el mundo y otro para la cámara.

de un punto y apunta a otro punto. La segunda, obtener_punto, permite obtener un punto situado sobre el rayo a la distancia especificada del origen de éste. La tercera, distancia, devuelve la distancia al origen del rayo de un punto situado sobre él.

LA CÁMARA COMO PROTAGONISTA

La cámara que se va a emplear para captar la escena está inspirada en las cámaras de fotos reales, pero con una diferencia: está al revés. En las reales, el agujero por donde entra la luz está situado delante de la película que capta la imagen. En ésta, la película (el monitor) está situada delante y el agujero (el punto de vista) detrás. Es como si se abriera una ventana al mundo virtual que se ha creado por la que se ven unos objetos u otros, dependiendo de la posición desde donde se mire. Por lo tanto, lo primero que hay que definir en la cámara es su posición, y hacia dónde apunta. Es decir, la situación de una cámara viene dada por un elemento de la clase rayo anteriormente definida.

Otro elemento importante es el número de píxels que va a tener la imagen. Cuantos más píxels tenga la imagen mayor calidad se obtendrá, aunque se seguirá viendo la misma escena. Es un concepto similar al de los ASA de las cámaras fotográficas: cuanto mayor es el ASA, mas fino es el "grano" de la fotografía obtenida y, por lo tanto, mayor calidad.

El último elemento que define una cámara es su ángulo focal, que está representado en la figura 3. Es el ángulo horizontal que forma el punto de observación con los extremos horizontales del plano de imagen. Cuanto más grande es el ángulo focal, menor es la distancia focal (distancia del punto de observación al plano de imagen), y se obtiene más campo de visión (se ve todo desde más lejos). Es el zoom de la cámara.

El color de un píxel depende de los rayos de luz que inciden sobre él

Una vez definidos todos estos elementos, se calcula la posición del centro de la pantalla en coordenadas del mundo. Para ello, se obtiene la distancia focal mediante la fórmula trigonométrica:

Seno (alfa) / Coseno (alfa) = cateto opuesto / cateto contiguo

donde alfa es el ángulo focal en radianes dividido entre 2, el cateto opuesto es el número de píxels en x dividido entre 2, y el cateto contiguo es la distancia focal a calcular. Todos estos elementos se aprecian en la figura 3. Despejando en la fórmula, se obtiene:

distancia = píxels_x * Coseno (alfa) / (2 * Seno (alfa))

El centro del plano de imagen se obtendrá mediante la función obtener_punto anteriormente comentada, pasándole como parámetro la distancia focal obtenida.

Por último, se definen los vectores horizontales y verticales del sistema de coordenadas de la cámara. El vector horizontal es paralelo al suelo y perpendicular al vector director del rayo de la cámara. Su cálculo es muy sencillo. Si el vector director es (x, y, z), el vector horizontal será (-z, 0, x). Pero hay una excepción. Si el vector es completamente vertical, tanto x como z son nulos y se obtendría el vector (0, 0, 0). En este caso, se toma por convenio el vector (1, 0, 0). El vector vertical se obtiene haciendo el producto vectorial de los vectores horizontal y director. Con estos vectores se puede saber las coordenadas en el mundo de cualquier píxel de la pantalla. Y, por lo tanto, se puede construir un rayo que parta del punto de observación y pase por cualquier píxel. Esto es lo que hace la función por_píxel.

LOS OBJETOS, ORIENTADOS A OBJETO

La clase objeto se va a utilizar para representar cualquier objeto en el mundo que se está construyendo. Es una clase genérica y, en realidad, sirve para definir los elementos básicos que debe poseer cualquier objeto. Estos elementos son:

- Su posición. Define las coordenadas del centroide del objeto.
- Sus materiales. Se pueden definir dos materiales diferentes y hacer patrones con ellos que se aplicarán sobre el objeto. Esto se explica un poco más adelante.
- Un puntero al siguiente objeto. Sirve para encadenar todos los objetos en forma de lista. Si el puntero es NULL, quiere decir que ya no hay ninguno más.
- Un puntero a una función de generación de patrón. También se explicará más adelante.
- Una función de intersección. Decide si el objeto está o no en intersección con un rayo. Si no es así, devuelve el valor -1. De lo contrario intersecciona, y devuelve la distancia del punto de intersección al origen del rayo. Es la función más importante y da toda la flexibilidad y potencia al sistema. Se podrá construir cualquier tipo de objeto e incorporarlo a la escena siempre que se sea capaz de definir esta función para ese objeto.

Cualquier objeto que se quiera añadir al mundo debe ser un descendiente de la clase objeto y, por lo tanto, tiene que

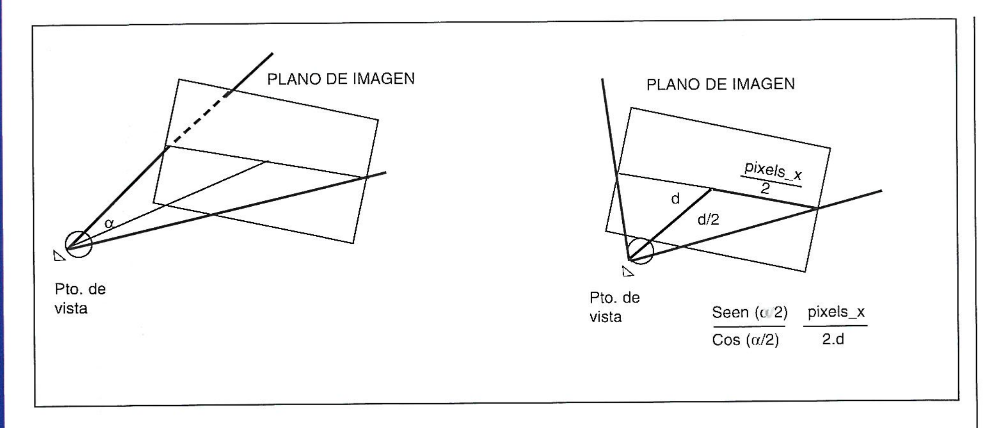


Figura 3. Cuanto más grande es el ángulo focal, más campo se abarca.

definirse su función de intersección con un rayo.

EL SUELO, EL EJEMPLO MÁS SENCILLO

En este mundo, el suelo es un objeto más. Como se ha dicho antes, lo único necesario será poder definir su intersección con un rayo.

Lo primero es saber cuándo un rayo va a terminar "chocando" contra el suelo. Es sencillo: cuando apunte hacia abajo. Aunque esto parece evidente, hay que recordar que un rayo tiene un punto de partida, y por lo tanto, sólo interesan aquellas intersecciones que se produzcan más allá del punto de partida, y no antes. También hay que tener en cuenta que ningún rayo va a proceder de debajo del suelo (al menos, de momento), ya que la cámara está situada sobre él.

Para saber si un rayo r apunta hacia abajo, no hay más que comprobar la componente Y de su vector director. Es decir, si

r.dir.y <= 0

el rayo hace intersección con el suelo. Para obtener la distancia a la que se produce la intersección, hay que tener en cuenta que

r.pos.y = - t * r.dir.y

Despejando t, se obtiene:

t = - r.pos.y / r.dir.y

Para evitar divisiones por números próximos al cero, se hace la división con el mínimo entre r.dir.y (que es negativo) y -EPSILON, que es una constante predefinida.

Hay un caso especial, que es el horizonte. Un rayo hace intersección con el horizonte cuando la componente Y de su vector director es 0, es decir, cuando es paralelo al suelo. Este caso queda contemplado en el párrafo anterior, ya que se terminaría dividiendo por -EPSILON.

LAS ESFERAS, MÁS DIFÍCIL TODAVÍA

Para poder incluir esferas en el mundo que se está creando, una vez más, hay que definir su intersección con un rayo. Una esfera queda definida por su posición o centro (xc, yc, zc) y su radio r. Todos los puntos de la superficie de la esfera cumplen que su distancia al centro al cuadrado es igual al radio al cuadrado. Es decir:

(x - xc)^2 + (y - yc)^2 + (z - zc)^2 = r^2

Como se ha dicho antes, un rayo viene definido por la expresión:

R(t) = Ro + Rd * t, con t > 0.

Ro es el punto de origen, definido como (xo, yo, zo). Rd es el vector director, definido como (xd, yd, zd). Sustituyendo en la expresión del rayo, se obtiene:

x = xo + xd * t
y = yo + yd * t
z = zo + zd * t

Si se sustituyen estas expresiones en la ecuación de la esfera, se obtiene:

(xo + xd * t - xc)^2 + (yo + yd * t - yc)^2 + (zo + zd * t - zc)^2 = r^2

Lo cual, es una ecuación de segundo grado cuya incógnita es t, que es la distancia desde el origen del rayo a la que se produce la intersección. Esta ecuación puede tener dos soluciones (el rayo atraviesa la esfera), una solución (el rayo es tangente a la esfera), o no tener solución (el rayo no hace intersección con la esfera). Además, sólo interesan las soluciones positivas, ya que las negativas indican que la intersección se ha producido antes del origen del rayo. Y eso no es todo, en caso de haber dos soluciones positivas, sólo interesará la menor de las dos, que es el punto en donde el rayo hace intersección antes.

Una ecuación de segundo grado se puede expresar como:

A * t^2 + B * t + C = 0

En este caso,

A = xd^2 + yd^2 + zd^2 = 1

B = 2 * (xd * (xo - xc) + yd * (yo - yc) + zd * (zo - zc))

C = (xo - xc)^2 + (yo - yc)^2 + (zo - zc)^2 - r^2

El término A siempre va a ser exactamente 1, ya que, como se ha dicho antes, todos los vectores directores están normalizados.

Para hacer más simples los términos B y C, se calcula un elemento auxiliar:

aux (xa, ya, za) = (xo - xc, yo - yc, zo - zc)

Aplicando esto en las expresiones, se obtiene:

A = 1
B = 2 * (aux | dir)
C = aux | aux - r^2

Hay que recordar que | es el operador que realiza el producto escalar de dos vectores.

Las soluciones de una ecuación de segundo grado vienen dadas por las expresiones:

t0 = (- B - sqrt (B^2 - 4 * A * C)) / (2 * A)

t1 = (- B + sqrt (B^2 - 4 * A * C)) / (2 * A)

La función de intersección debe ser lo más rápida posible, por lo que antes de hacer la raíz cuadrada, se calcula su discriminante (lo que está dentro de sqrt ()). Si este es menor que cero no hace falta hacer la raíz cuadrada, ya que ésta no existe y por lo tanto, no hay soluciones a la ecuación (ni intersecciones). Si el discriminante es cero, tampoco hará falta hacer la raíz, y se considerará que tampoco hay intersección (aunque en realidad hay un punto, se hace así por acelerar la función). Si el discriminante es mayor que cero, primero se calcula t0, que siempre será menor que t1. Si t0 es mayor que 0, es la intersección buscada. Si no, se devuelve el valor de t1 (que también puede ser negativo).

Una vez explicada paso a paso esta función de intersección, sería interesante que el lector tratara de programar las intersecciones con planos, triángulos,

conos, cilindros, toros, etc., o con sus propios objetos, e incorporarlos a este mundo que se está creando. ¡Ánimo, no es tan complicado!

LOS MATERIALES SON LA ESENCIA

Bueno, ya se ha definido prácticamente todo lo necesario para empezar a trazar rayos. Sólo falta algo que va a ser esencial: los materiales. Un material es una clase, que de momento va a ser realmente simple. El único elemento que hay definido es el coeficiente de luz difusa.

El coeficiente de luz difusa mide la cantidad de luz que es absorbida por un objeto. O dicho de otra forma, es el color del objeto. Todo color puede ser definido por sus tres componentes rojo, verde y azul, que a partir de ahora se denominarán r, g y b respectivamente. Por lo tanto, se precisará un elemento de la clase vector para definir este coeficiente. Los valores de cada componente deben estar comprendidos entre 0 y 1. De esta forma, si el coeficiente de luz difusa de un objeto es (1, 0, 0), éste será de color rojo puro. Si su coeficiente es (0, 0, 0.3), el objeto será de color azul obscuro.

La clase material irá creciendo capítulo a capítulo, por lo que los objetos que se consigan serán cada vez más reales.

DOS SON COMPAÑÍA: LOS PATRONES

La clase objeto que se ha definido está formada, entre otras cosas, por dos materiales. Estos dos materiales van a servir para definir, si se desea, un patrón. Un patrón es una combinación cualquiera de dos materiales. En realidad, pueden definirse patrones con tantos materiales como se desee. Lo único necesario es definir una función que decida para un punto dado qué material le corresponde. Los objetos tienen un puntero a una función del tipo:

```
int patron (vector punto)
```

Por lo tanto, para definir un patrón no hay más que definir los dos materiales, programar esta función y hacer que el puntero del objeto apunte hacia ésta. En el programa se han incluido dos ejemplos muy simples de patrones: uno para el suelo y el otro para las franjas de una de las esferas.

EL MUNDO, LA COMBINACIÓN DE TODO

La clase mundo va a reunir todos los elementos definidos anteriormente. Es decir, un mundo estará formado por una cámara y una lista de objetos. Así de simple. Adicionalmente, se ha incluido un color de cielo, para cuando el rayo no hace intersección con ningún objeto.

El constructor de la clase mundo que se ha programado crea la cámara, tres esferas y un suelo y los incorpora al mundo "a capón". Esto quiere decir que no hay posibilidad de cambiar la configuración del mundo desde fuera del programa (de momento). Para crear una nueva escena es necesario recompilar la aplicación. Se deja en manos del lector escribir una función que lea de un fichero ASCII los elementos que componen la escena a visualizar, definiéndose previamente un lenguaje de descripción para este fichero. Este tipo de función recibe el nombre de parser, y es un buen desafío a todo programador que se precie. No ha sido incluido en este capítulo por no hacerlo más complicado.

La función más interesante de esta clase se llama trazar y recibe como parámetros las coordenadas de un píxel y devuelve el color de éste en la "foto" que se está haciendo de la escena. Esta función crea un rayo que parte del observador y pasa por el píxel. Busca las intersecciones de este rayo con todos los objetos que componen la escena, quedándose con el objeto cuya intersección es la más próxima de todas las encontradas. La función devuelve el color de este objeto o, si no encuentra ninguna intersección, el color definido para el cielo.

¿Y QUÉ SE HACE CON TANTO PÍXEL?

Ya está todo definido y entendido (¿no?). Lo único que falta es poder guardar la imagen que se obtenga en un fichero. Para ello, se ha programado la clase targa, que permite crear un fichero gráfico targa. Este formato gráfico permite crear imágenes con 24 bits de color (un byte para el rojo, otro para el verde y otro para el azul) por cada píxel. Su estructura es realmente simple. Para más información, se puede consultar el curso de formatos gráficos publicado en esta revista.

BIBLIOGRAFÍA

- "An Introduction to ray tracing". Editado por Andrew S. Glassner. Editorial Academic Press.
- "Fractal programming and ray tracing with C++". Roger T. Stevens. Editorial M&T Publishing, Inc.
- "Computer Graphics. Principles and practice". Foley, van Dam, Freiner y Hughes. Editorial Addison Wesley.

Hay tres funciones para esta clase:

- Un constructor, al que se le pasa el nombre del fichero y las dimensiones de la imagen.
- La función anotar, que escribe secuencialmente un píxel en el fichero. Puesto que la escritura es secuencial, los píxels deben ser obtenidos secuencialmente, esto es, de izquierda a derecha y de arriba a abajo.
- Un destructor, que termina de guardar la imagen y cierra el fichero.

Y... ¡ACCIÓN! ... ¿ACCIÓN?

El programa principal es realmente simple: llama al constructor del mundo, al constructor del fichero tga, hace dos bucles anidados para recorrer todos los píxels, traza un rayo por cada píxel, escribe el resultado en el fichero y llama a los destructores. ¡Más fácil imposible!

Después de tanta clase y tanta geometría, quizá el lector quede un poco defraudado con los resultados obtenidos. No sólo tendrá que esperar varios minutos para generar la imagen (o varias horas, si no tiene un 486 o un 386 con coprocesador matemático), si no que la imagen que obtendrá no es más que tres círculos de color uniforme, y un horroroso suelo de losetas que en el horizonte parece una carrera de hormigas. ¿Eso es todo?

Bueno, la verdad es que el enorme tiempo de generación de la imagen es el carísimo precio que hay que pagar en el afamado "club ray tracing". Aunque más adelante se verán técnicas de aceleración, no va a haber mejoras espectaculares.

Lo que sí va a cambiar completamente es el resultado obtenido. En cuanto se añadan luces, colores degradados, reflexiones, refracciones, luces especulares y texturas, podemos asegurarle que las imágenes obtenidas van a ser increíblemente reales y espectaculares.

ICONOS Y CURSORES

Jorge R. Regidor

Se sabe que todas las culturas tienen una serie de signos o símbolos, que instintivamente y para todos los miembros del grupo, informan sobre diversas situaciones y acciones. Por poner un ejemplo sencillo y orientado al lenguaje simbólico mundialmente más extendido, el de tráfico; al observar un semáforo en rojo sabemos que no debemos pasar, ya que ese semáforo nos indica o informa de la posibilidad de que existan coches cruzando. En Windows, los iconos forman el lenguaje simbólico del sistema, indicando según las diferentes situaciones peligro (al formatear un disco), catástrofe (error en el fichero que en ese momento acabábamos de grabar), etc.

Por otra parte y aunque anteriormente su uso no era ese, los cursores al ser también iconos, a parte de apuntar y servir para abrir, ejecutar,... programas, nos informan sobre el estado del programa, por ejemplo si está cargando datos de disco y se debe esperar, el icono cambia a un reloj que simboliza dicha espera.

Sin más preámbulos, vamos a iniciar el modo de crear y utilizar tanto iconos como cursores. Es un tema realmente sencillo y que sin embargo, y en el caso de estar bien implementado puede darle un aire totalmente diferente a nuestros programas.

ICONOS

Un icono se puede definir como una o varias imágenes de unas dimensiones y números de colores establecidas en función de las capacidades gráficas del sistema. De esta forma un icono puede tener imágenes para una tarjeta CGA, EGA, VGA y SVGA; la tabla A muestra el tamaño y formato de colores en función del sistema de gráficos. Además cada imagen consta de dos bitmaps para conseguir el efecto de transparencia. Toda esta información va empaquetada en una estructura y guardada sobre ficheros de extensión .ICO.

El uso habitual de los iconos pasa por que éstos sean el icono del programa, desde el cual se ejecuta, o bien sirvan para ilustrar o informar dentro del programa.

ANCHO	ALTO	COLORES	TARJETA
16	16	2	CGA
32	32	2	HERCULES
32	32	8	EGA
32	32	16	VGA y 8514/A
64	64	256	SVGA (Alta Res.)

Tabla A. Tamaño y colores de los diferentes formatos en un icono.

ESTABLECER EL ICONO DEL PROGRAMA

Para establecer el icono del programa, se debe cargar éste como parte de la información suministrada a Windows cuando se registra la clase de la ventana principal. Los iconos al igual que los demás recursos se enlazan a la aplicación en un segmento a propósito para ellos. La forma de cargar un recurso desde el código fuente es mediante la función LoadIcon, cuyo formato es el siguiente:

En el entorno gráfico de Windows, tanto iconos como cursores son recursos ampliamente usados por todas las aplicaciones, residiendo su utilidad en resaltar algún tipo de acción o bien un estado del programa. Además, como todos sabemos los cursores son la extensión de la mano a través del ratón para abrir, cerrar, mover... objetos y programas.

HICON LoadIcon(hInst, pszIcon)

donde,

hInst es el handle a la instancia de la cual queremos cargar los recursos, normalmente la instancia de nuestro programa. Si se le pasa un valor NULL, los recursos serán cargados del sistema, donde hay definidos unos iconos standard por defecto que son los indicados en la tabla B.

pszIcon es una cadena de caracteres que contiene el nombre asignado al icono. En el caso de no haber asignado un nombre al icono sino un número o identificador, se debe utilizar la macro MAKEINTRESOURCE, para cargar el icono, su formato es:

LoadIcon(hInst, MAKEINTRESOURCE(IDI_ICON));

donde,

DC_ICON es un número entero, identificador asignado en Workshop o AppStudio.

La función LoadIcon devuelve un handle del tipo HICON, al icono cargado.

El siguiente listado ilustra el modo de asignar el icono al programa, asignando sobre el campo hIcon de la estructura WNDCLASS el icono que se desea tener.

```
WNDCLASS wndMiWindow;
 wndMiWindow.style        = CS_VREDRAW | CS_HREDRAW;
 wndMiWindow.lpfnWndProc  = (WNDPROC)wndProcWnd;
 wndMiWindow.cbClsExtra   = 0;
 wndMiWindow.cbWndExtra   = 0;
 wndMiWindow.hInstance    = hInst;
 wndMiWindow.hIcon        = LoadIcon(hInst, "ICONO");
 wndMiWindow.hCursor      = NULL;
 wndMiWindow.hbrBackground = GetStockObject(WHITE_BRUSH);
 wndMiWindow.lpszMenuName = NULL;
 wndMiWindow.lpszClassName = "MiWindow";
 RegisterClass(&wndMiWindow);
```

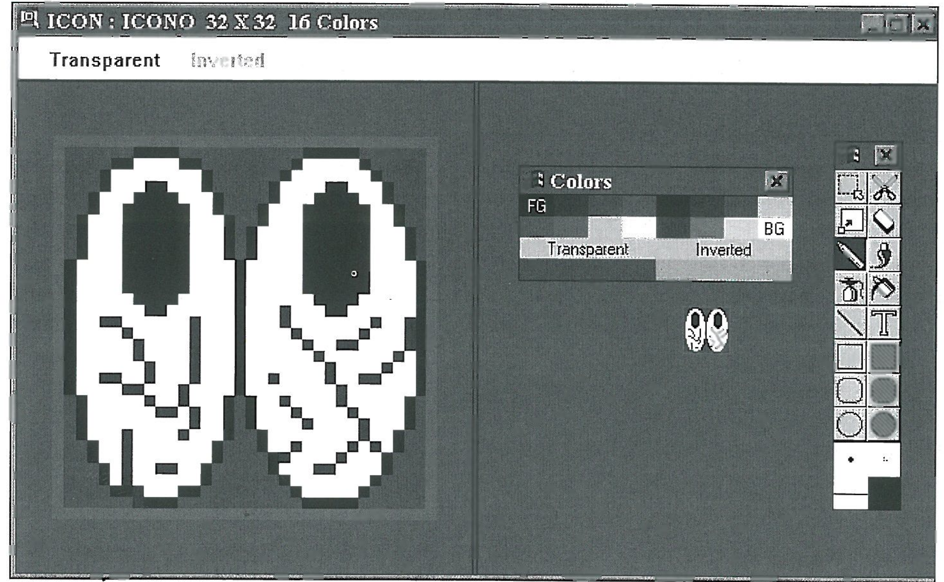


Figura A. Creación de un icono desde Workshop.

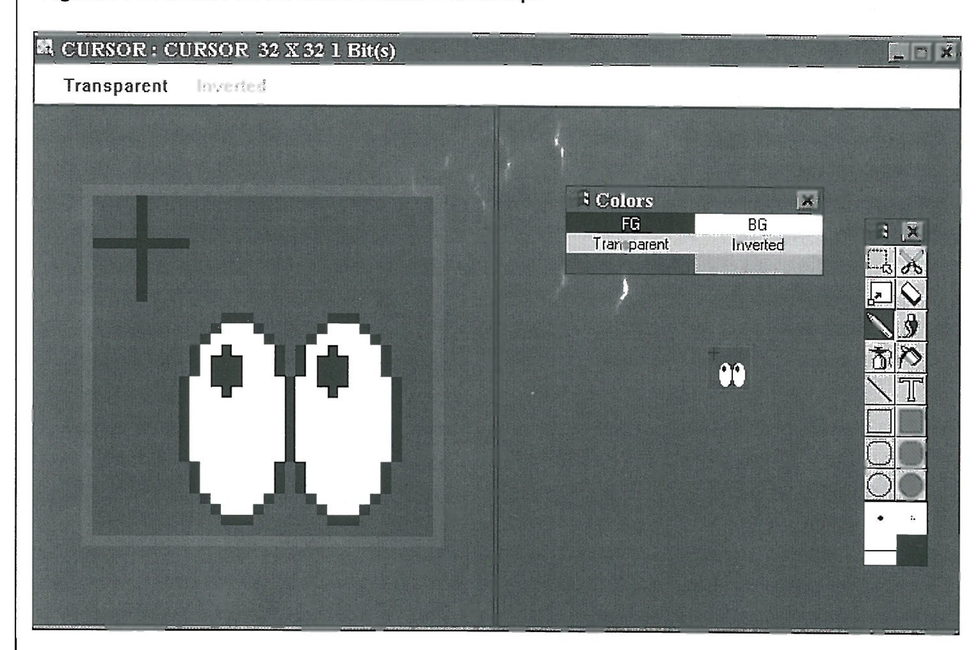


Figura B. Creación de un cursor desde Workshop.

Como se puede observar en este ejemplo se carga el icono con nombre ICONO como icono del programa.

ICONOS MOSTRADOS DENTRO DE UNA VENTANA

El otro uso de los iconos, mostrándolos dentro de una ventana, se realiza cargando el icono con LoadIcon, y se pinta sobre el contexto de dispositivo de la ventana. La mejor forma de seguir el proceso pasa por poner un ejemplo sencillo del proceso y explicar cada una de las funciones. El listado para este ejemplo es el siguiente:

```
case WM_PAINT:
```

ICONOS	*SIGNIFICADO*
IDI_APLICATION	Icono por defecto por las aplicaciones.
IDI_ASTERISK	Indica información.
IDI_EXCLAMATION	Indica advertencia.
IDI_HAND	Para situaciones peligrosas.
IDI_QUESTION	Para situaciones con toma de decisiones.

Tabla B. Iconos estándar de Windows.

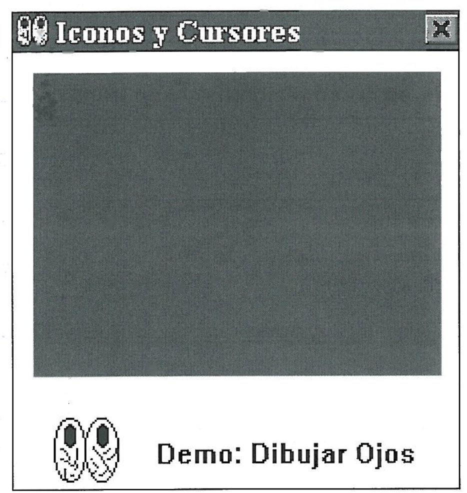


Figura C. Programa ejemplo de iconos y cursores.

```
{
  HDC hDC;
  PAINTSTRUCT psPaint;
  HICON hIcon;
  hDC = BeginPaint(hWnd, &psPaint);

  hIcon = LoadIcon(hInst,"ICONO");
  DrawIcon(hDC, 10, 20,hIcon);
  DestroyIcon(hIcon);

  EndPaint(hWnd, &psPaint);
  return 0;
}
```

Como se observa en el listado, parte de la atención de un mensaje WM_PAINT, proceso lógico, ya que si todo lo que se pinte en una ventana y se quiera mantener al minimizar la ventana o haber sido solapada, debe ser pintada como respuesta a este mensaje. Se declaran tres variables locales necesarias para el proceso, HDC es un handle al contexto de pantalla, PAINTSTRUCT se usa para obtener la información necesaria para repintar la ventana y por último HICON necesaria para cargar el icono. La primera función llamada es BeginPaint, que no es objeto de este artículo, pero debemos saber que nos devuelve el handle al contexto de pantalla y llena la estructura PAINTSTRUCT. La siguiente línea se encarga de cargar el icono, y DrawIcon pinta el icono cargado sobre hIcon en el contexto de pantalla de la ventana. El formato de DrawIcon es:

En Windows, los iconos forman el lenguaje simbólico del sistema

BOOL DrawIcon(hDC, x, y, hIcon)

donde,

hDC es el contexto de pantalla.
x es la coordenada x,dentro del contexto, donde se pinta el icono.
y es la coordenada y, dentro del contexto, donde se pinta el icono.
hIcon es el handle del icono cargado con LoadIcon.

devuelve un valor distinto de cero si ha sido pintado el icono correctamente.

Una vez pintado el icono, se debe liberar la memoria del icono con la función DestroyIcon cuyo formato es:

BOOL DestroyIcon(hIcon)

donde,

hIcon es el handle del icono cargado con LoadIcon.

La función devuelve un valor distinto de cero si el icono ha sido debidamente destruido.

Después de este proceso, y para visualizar en pantalla los cambios se llama a la función EndPaint, pasándole el handle de la ventana, y un puntero a la estructura PAINTSTRUCT obtenida con BeginPaint.

Si este proceso se realiza fuera de un mensaje WM_PAINT, la única variante radica en la forma de obtener el handle al contexto de pantalla. El siguiente listado muestra la forma de realizar dicho proceso.

```
HDC hDC;
POINT ptMouse;
HICON hIcon;

hDC = GetDC(hWnd);

hIcon = LoadIcon(hInst,"ICONO");
DrawIcon(hDC, 10, 20, hIcon);
DestroyIcon(hIcon);

ReleaseDC(hWnd, hDC);
```

Como se puede apreciar el handle al contexto de dispositivo se obtiene mediante la función GetDC y se libera con la función ReleaseDC.

CURSORES

Los cursores son iconos especialmente dedicados a la labor de apuntar sobre los objetos del sistema. Generalmente tienen un tamaño de 32x32 pixels y dos colores, aunque pueden llegar a ser de hasta 64x64 pixels y 256 colores. Al igual que los iconos, puede existir sobre un mismo fichero (.CUR) varias imágenes del cursor para los diferentes modos gráficos.

Existen dos formas de asociar un cursor con una ventana, asignándolo a toda las ventanas de la clase a la que pertenece o sólo a una ventana.

ASIGNAR UN CURSOR A UNA CLASE DE VENTANA

Al igual que con los iconos, este proceso se realiza cargando sobre el campo hCursor de la estructura WNDCLASS el cursor que queremos. El siguiente listado muestra el modo de hacerlo.

```
WNDCLASS wndMiWindow;

wndMiWindow.style         = CS_VREDRAW | CS_HREDRAW;
wndMiWindow.lpfnWndProc   = (WNDPROC)wndProcWnd;
wndMiWindow.cbClsExtra    = 0;
wndMiWindow.cbWndExtra    = 0;
wndMiWindow.hInstance     = hInst;
wndMiWindow.hIcon         = LoadIcon(hInst, "ICONO");
wndMiWindow.hCursor       = LoadCursor(NULL, IDC_ARROW);
wndMiWindow.hbrBackground = GetStockObject(WHITE_BRUSH);
wndMiWindow.lpszMenuName  = NULL;
wndMiWindow.lpszClassName = "MiWindow";

RegisterClass(&wndMiWindow);
```

La función encargada de cargar el cursor desde los recursos es:

CURSORES	UTILIDAD/FORMA
IDC_ARROW	Flecha estandar.
IDC_CROSS	Cursor en forma de cruz.
IDC_IBEAM	Forma de I, para controles de edicion.
IDC_ICON	Ventana pequeña.
IDC_SIZE	Cursor para mover una ventana.
IDC_SIZENESW	Cursor para estirar una ventana.
IDC_SIZENS	Cursor para estirar una ventana.
IDC_SIZENWSE	Cursor para estirar una ventana.
IDC_SIZEWE	Cursor para estirar una ventana.
IDC_UPARROW	Flecha vertical.
IDC_WAIT	Reloj de arena que simboliza espera.

Tabla C. Cursores estándar de Windows.

HCURSOR LoadCursor(hInst, pszCursor)

donde,

hInst es la instancia donde están los recursos. Normalmente, la instancia del programa. Si le pasa un valor NULL, se puede cargar uno de los cursores definidos por el sistema. La tabla C muestra los posibles valores de dichos cursores.

pszCursor es el nombre del cursor en el fichero de recursos. Si está definido con identificador hay que utilizar MAKEINTRESOURCE.

La función devuelve el handle del tipo HCURSOR al cursor cargado.

ESTABLECER EL CURSOR DE UNA VENTANA

Si se necesita que dentro de una clase de ventanas no todas tengan el mismo cursor, existe un método para poder conseguirlo. Primero hay que especificar en el campo hCursor de la estructura WNDCLASS de la clase de ventanas, un valor NULL.

wndMiWindow.hCursor = NULL;

Luego, hay que incluir el siguiente código:

```
case WM_CREATE:
{
        SetCursor(LoadCursor(NULL, IDC_CROSS));

    .... nuestro código ....

    return 0;
}
case WM_MOUSEMOVE:
{
        SetCursor(LoadCursor(NULL, IDC_CROSS));

    .... nuestro código ....

    return 0;
}
```

Con la función SetCursor, se asigna un cursor a la ventana actual, pero si vamos a otra y luego volvemos el cursor no se mantiene. Por eso, se añade esta línea también en el mensaje WM_MOUSEMOVE, puesto que al entrar en la ventana el ratón se mueve y se asigna de nuevo el cursor.

Existen dos formas de asociar un cursor con una ventana, asignándolo a toda las ventanas de la clase a la que pertenece, o sólo a una ventana

CAMBIAR EL CURSOR DE UNA CLASE DE VENTANAS

Para cambiar el cursor de una clase de ventanas después de su registro, hay que utilizar una función que sirve para cambiar las propiedades de una clase de ventanas. Esta función SetClassWord, cuyo formato para el caso de cambiar el cursor es:

SetClassWord(hWnd, GWC_HCURSOR, hCursor);

donde,

hWnd es el handle de una ventana perteneciente a la clase que se quiere cambiar.

GWC_HCURSOR es una constante que indica que se quiere cambiar el handle del cursor.

hCursor es el handle del cursor cargado con LoadCursor.

Para cambiar el icono de programa también se puede utilizar esta función con el siguiente formato:

SetClassWord(hWnd, GWC_HICON, hIcon);

EL PROGRAMA EJEMPLO

En este programa se ilustran los métodos principales en cuanto a lo que la programación de cursores e iconos se refiere. El programa al iniciarse muestra una ventana con un rectángulo gris. En la zona blanca, el cursor es una cruz y se asigna con el método de SetCursor. En la zona gris y al pulsar el botón derecho el cursor cambia con SetClassWord. En este punto, si se pulsa el botón izquierdo del ratón, se pinta en la posición actual el icono con la imagen de los ojos.

Sobre el código fuente hay que resaltar la función que se encarga de limitar el movimiento del ratón sobre una zona de la pantalla. Dicha función se utiliza al pulsar el botón derecho del ratón sobre el cuadro gris, (al atender a WM_RBUTTONDOWN de la función wndProcCursorWindow) y su formato es:

ClipCursor(lprc)

donde,

lprc es un puntero FAR a una estructura RECT que contiene las coordenadas del rectángulo el cual va a limitar el movimiento al ratón. Si se le pasa un valor NULL el resultado es que se libera al ratón para moverse libremente.

CREACIÓN DE UN MENÚ DE DESARROLLADOR

José María Peco

En el entorno DIGITAL, se trabaja de una forma muy parecida a como se trabaja en un PC. De entrada, cuando se conecta un usuario, el sistema siempre le asigna un directorio por defecto, que suele tener el nombre lógico sys$login, y además, ejecuta los procedimientos de conexión que cargan todas aquellas variables necesarias para el entorno. Pero, lo mas importante, por lo que respecta al tema que nos ocupa, es que termina ejecutando el fichero de comandos LOGIN.COM, en el caso de que exista, del directorio asignado por defecto. Este fichero, que equivale al tan conocido AUTOEXEC.BAT del MS-DOS, es el que normalmente usaremos para realizar aquellas asignaciones necesarias para poder personalizar nuestra gestión de desarrollo. Como ejemplo se muestra en la figura 1 el listado del archivo LOGIN.COM.

En Digital, los procedimientos de comandos pueden escribirse tanto en minúsculas como en mayúsculas, pues el procesador internamente las trata todas como mayúsculas. No ocurre lo mismo con los valores asignados a las variables, pues diferencia unas de otras, debiendo considerar esta circunstancia para examinar la opción elegida por el usuario en un menú.

Por lo que respecta al fichero citado cabe hacer los siguientes comentarios:

Por ser un archivo de comandos, cada uno de los debe comenzar por el símbolo $

Los comentarios se identifican por el símbolo '!' por lo que las tres primeras líneas lo son.

La 4ª línea especifica, que cuando se invoque este procedimiento en un modo distinto al interactivo, es decir en batch, se salga de él.

La siguiente línea programa la tecla E2 (Insertar) con el valor que se indica, y que se corresponde con la ejecución del fichero de comandos MENU. El hecho de especificar el direccionamiento completo de dicho archivo, es para poder ejecutarle con independencia del directorio en el que se encuentre el desarrollador. Es de resaltar que como no cabe todo el comando en una línea, se utilizan dos, razón por la cual, la primera termina con un guión '-' y la siguiente empieza sin el '$'.

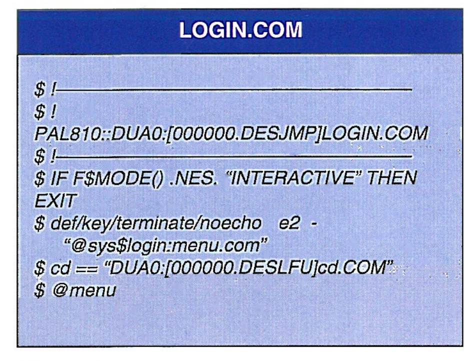
La siguiente línea asigna al símbolo CD el direccionamiento completo de otro archivo de comandos, que permitirá emular el comando CD del entorno MS-DOS.

Por ultimo, se ejecuta el archivo de comandos MENU.COM (pues va precedido del símbolo @ y al no especificar extensión, asume que es .COM).

Este último archivo, MENU.COM es el objetivo de este artículo, por lo que se

El fichero LOGIN.COM equivale al conocido AUTOEXEC.BAT del MS-DOS

El pasado mes se inició este tema hablando de la creación de un menú o panel de selección en el entorno de IBM. Este mes se continúa el tema, pero para hablar de otro entorno diferente, el entorno DIGITAL. En cualquier caso, sólo se presentan distintas formas o funciones básicas que pueden ser utilizadas por el programador para personalizar su propia gestión de desarrollo.



LOGIN.COM

```
$ !——————————————————
$ !
PAL810::DUA0:[000000.DESJMP]LOGIN.COM
$ !——————————————————
$ IF F$MODE() .NES. "INTERACTIVE" THEN
EXIT
$ def/key/terminate/noecho   e2  -
     "@sys$login:menu.com"
$ cd == "DUA0:[000000.DESLFU]cd.COM"
$ @menu
```

Figura 1.

expone a continuación un menú muy reducido que permita iniciarse en esta función, si bien en próximos artículos se ira ampliando con nuevas opciones el menú que hoy se comenta. Cabe resaltar, eso sí, antes de empezar, que los números que figuran a la izquierda sólo son una referencia para poder comentar cada una de las líneas, por lo que aquellos que quieran ejecutar este ejemplo, deberán hacerlo sin escribir dicho número.

El listado completo se muestra en la figura 2, y los comentarios a cada una de las líneas es el siguiente:

007 asigna a la variable esc[0,8] el valor 27 ascii, es decir el carácter ESCAPE.

008 carga la variable WS para utilizarla como abreviatura de write SYS$output (es decir escribe sobre donde esté definida la salida lógica de mensajes).

009 asigna a KEY_NUM los caracteres que activa el teclado numérico.

010 asigna a la variable CLS los caracteres para borrar la pantalla.

011 establece el prompt del usuario como JMP> (Lo lógico sería incluir esta sentencia en el LOGIN.COM).

012 hace que el usuario se posicione en el directorio que tiene definido por defecto. De esta forma regresa a él cuando, al pulsar la tecla E2 programada en LOGIN.COM, se ejecute este procedimiento.

023 y 024 activa teclado numérico y borra pantalla.

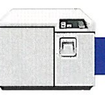
025 asigna a la variable CABECERA el valor especificado, donde 0;35H indica que escriba en la fila 0 columna 35. #6 indica que escriba en modo de 40 columnas por línea; y 0m indica que escriba en intensificado, lo que sigue hasta encontrar las dobles comillas (JMP).

MENU.COM

```
001 $ IF F$MODE() .NES. "INTERACTIVE"
THEN EXIT
002 $ !
=======================================
===========================
003 $ !                 Menu de UTILIDADES de
JMP
004 $ !
=======================================
===========================
005 $ !
006 $ !—————————— asignaciones
007 $   esc[0,8]  = 27
008 $   ws        = "write sys$output"
009 $   key_num   = ""esc'>"
010 $   cls       = ""esc'[2J"
011 $   set prompt="Jmp> "
012 $   set def sys$login
020 $ !——————————————————
021 $ menu:
022 $ !
023 $   ws key_num
024 $   ws cls
025 $   CABECERA =
""esc'[0;35H"esc'[J"esc'#6"esc'[0m  JMP "
027 $   ws ""CABECERA""
028 $ ! =============== definicion de la pan-
talla ======================
029 $   ws ""esc'[04;10H"esc[1;7m ** Directorio
**"esc'[0m"
031 $   ws ""esc'[06;10H"esc'[1m1."esc'[0m
Usuario       "esc'[0m"
032 $   ws ""esc'[07;10H"esc'[1m2."esc'[0m
Proyecto      "esc'[0m"
039 $   ws ""esc'[19;05H"esc'[1m8."esc'[0m 80
columnas   "esc'[0m"
040 $   ws ""esc'[20;05H"esc'[1m9."esc'[0m 132
columnas  "esc'[0m"
041 $ !
042 $   ws ""esc'[04;45H"esc'[1;7m
Varios"esc'[0m"
045 $   ws ""esc'[07;45H"esc'[1mB."esc'[0m
Basic         "esc'[0m"
048 $   ws ""esc'[10;45H"esc'[1mE."esc'[0m
Editar        "esc'[0m"
041 $ !
053 $   ws
""esc'[22;05H"esc'[1;5m"mensaje""esc'[0m"
041 $ !
054 $ ! ======= fin de la definicion de la panta-
lla ===================
056 $   read sys$command -
          opc/prompt=""esc'[22;60HTeclee opciin:
"
057 $   opc = f$edit(opc,"TRIM, UPCASE")
058 $ !————————————————— examinar
entrada
059 $   if f$length(opc) .GT. 1
060 $      then mensaje = "Identifique la opci¢n
con un solo d¡gito"
062 $           goto menu
063 $   endif
064 $ !————————————————— examinar
opcion elegida
065 $   opciones = "1289BE"
066 $   encontrado = f$locate(opc,opciones)
067 $   if encontrado .eq. f$length(opciones)
068 $      then mensaje = "Seleccione una opci¢n
existente"
069 $           goto menu
070 $   endif
071 $   if opc .eqs. "1"
072 $      then set def sys$login
073 $           goto fin
074 $   endif
075 $   if opc .eqs. "2"
076 $      then set def desarrollo$disk:[proyecto]
077 $           goto fin
078 $   endif
079 $   if opc .eqs. "3"
080 $      then sh def exit
080 $           exit
081 $   endif
086 $   if opc .eqs. "8" then set terminal/width=80
087 $   if opc .eqs. "9" then set
terminal/width=132
088 $   if opc .eqs. "B"
089 $      then define sys$input
sys$command/nolog
090 $           basic
091 $   endif
092 $   if opc .eqs. "E"
093 $      then INQUIRE fichero "Archivo a editar
.. :"
094 $           define sys$input
sys$command/nolog
095 $           edit 'fichero
096 $   endif
097 $   if opc .eqs. "P"
098 $      then define sys$input
sys$command/nolog
099 $           phone
100 $   endif
101 $   if opc .eqs. "Q"
102 $      then define sys$input
sys$command/nolog
103 $           phone answer
104 $   endif
064 $ !——————————————————————

105 $ fin:
106 $   ws cls
107 $   exit
064 $ !——————————————————————
```


```
001 $ IF F$MODE() .NES. "INTERACTIVE"
THEN EXIT
002 $ !
=======================================
===========================
003 $ !                 Menu de UTILIDADES de
JMP
004 $ !
=======================================
===========================
005 $ !
006 $ !—————————— asignaciones
007 $   esc[0,8]  = 27
008 $   ws        = "write sys$output"
009 $   key_num   = ""esc'>"
010 $   cls       = ""esc'[2J"
011 $   set prompt="Jmp> "
012 $   set def sys$login
020 $ !——————————————————
021 $ menu:
022 $ !
023 $   ws key_num
024 $   ws cls
025 $   CABECERA =
""esc'[0;35H"esc'[J"esc'#6"esc'[0m  JMP "
027 $   ws ""CABECERA""
028 $ ! =============== definicion de la pan-
talla ======================
029 $   ws ""esc'[04;10H"esc[1;7m ** Directorio
**"esc'[0m"
031 $   ws ""esc'[06;10H"esc'[1m1."esc'[0m
Usuario       "esc'[0m"
032 $   ws ""esc'[07;10H"esc'[1m2."esc'[0m
Proyecto      "esc'[0m"
039 $   ws ""esc'[19;05H"esc'[1m8."esc'[0m 80
columnas   "esc'[0m"
040 $   ws ""esc'[20;05H"esc'[1m9."esc'[0m 132
```




```
'columnas "esc'[0m"
041 $ !
042 $  ws ""esc'[04;45H"esc'[1;7m
Varios"esc'[0m"
045 $  ws ""esc'[07;45H"esc'[1mB."esc'[0m
Basic        "esc'[0m"
048 $  ws ""esc'[10;45H"esc'[1mE."esc'[0m
Editar       "esc'[0m"
041 $ !
053 $  ws
""esc'[22;05H"esc'[1;5m"mensaje'"esc'[0m"
041 $ !
054 $ ! ======= fin de la definicion de la panta-
lla ===================
056 $   read sys$command -
          opc/prompt=""esc'[22;60HTeclee opciin:
"
057 $   opc = f$edit(opc,"TRIM, UPCASE")
058 $ ! ———————————— examinar
entrada
059 $   if f$length(opc) .GT. 1
060 $      then mensaje = "Identifique la opci¢n
con un solo d¡gito"
062 $           goto menu
063 $   endif
064 $ ! ———————————— examinar
opcion elegida
065 $   opciones = "1289BE"
066 $   encontrado = f$locate(opc,opciones)
067 $   if encontrado .eq. f$length(opciones)
068 $      then mensaje = "Seleccione una opci¢n
existente"
069 $           goto menu
070 $   endif
071 $   if opc .eqs. "1"
072 $      then set def sys$login
073 $           goto fin
074 $   endif
075 $   if opc .eqs. "2"
076 $      then set def desarrollo$disk:[proyecto]
077 $           goto fin
078 $   endif
079 $   if opc .eqs. "3"
080 $      then sh def exit
080 $           exit
081 $   endif
086 $   if opc .eqs. "8" then set terminal/width=80
087 $   if opc .eqs. "9" then set
terminal/width=132
088 $   if opc .eqs. "B"
089 $      then define sys$input
sys$command/nolog
090 $           basic
091 $   endif
092 $   if opc .eqs. "E"
093 $      then INQUIRE fichero "Archivo a editar
...:"
094 $           define sys$input
sys$command/nolog
095 $           edit 'fichero
096 $   endif
097 $   if opc eqs. "P"
098 $      then define sys$input
sys$command/nolog
099 $           phone
100 $   endif
101 $   if opc .eqs. "Q"
102 $      then define sys$input
sys$command/nolog
103 $           phone answer
104 $   endif
064 $ !
105 $ fin:
106 $   ws cls
107 $   exit
064 $ ! ————————————
```

Figura 2.

027 escribe en pantalla la variable CABECERA.

029 escribe en fila4 columna 10 y en vídeo inverso ([7m) el texto "directorio".

031 escribe en 7,10 un 1 en intensificado ([1m) y la palabra usuario como texto normal ([0m).

Así se escriben el resto de opciones, combinando la posición, con el tipo de atributo (intensificado, normal y vídeo inverso).

053 escribe en la posición 22,5 el contenido de la variable MENSAJE en parpadeante ([5m).

056 lee desde donde este definida la entrada de comandos (sys$command) es decir desde teclado, la opción que se elija, cargando la variable OPC.

057 transforma a mayúsculas el contenido de OPC.

059 examina la longitud de OPC y si es mayor que 1, carga la variable mensaje, cediendo el control al párrafo etiquetado con menu (línea 21) para revisualizar el panel de opciones y listar así el contenido del mensaje en parpadeante.

066 examina si la opción elegida se encuentra entre los caracteres que contiene la variable opciones, cargada en la línea anterior.

071 si la opción elegida es la 1, establece que el directorio por defecto sea SYS$LOGIN.

075 si la opción elegida es la 2, establece que el directorio por defecto sea el especificado.

086 si la opción elegida es la 8, establece que el terminal presente la información en 80 columnas.

087 ídem, pero con 132 columnas.

089 si la opción elegida es la B, define que la entrada del sistema sea la entrada de comandos, ya que de no hacer esta asignación, el sistema asume que los datos de entrada del programa que se va a ejecutar a continuación, recibe los datos desde el propio fichero de comandos.

090 invoca al programa BASIC.

093 pide el nombre del fichero que se desea editar. Esta es otra forma de pedir datos al usuario distinta a la vista en la línea 56, pero con la misma funcionalidad.

099 invoca al programa de utilidad PHONE normalmente para ejecutar luego DIAL id_usuario y poder llamar a otro usuario, o help, etc.

103 invoca al programa de utilidad PHONE pero con el parámetro ANSWER para poder responder a la llamada que se esta recibiendo.

106 limpia pantalla pues escribe (WS) el contenido de la variable CLS.

107 sale del procedimiento.

DESARROLLO DE PROGRAMAS C

David Aparicio

En la operativa cotidiana con Linux, nos encontramos a menudo con la necesidad de obtener los programas a partir de sus fuentes. Este mes, revisaremos las herramientas necesarias para ello.

El lenguaje de programación por excelencia en Linux, al igual que en prácticamente el resto de sistemas Unix, es C. Hasta que otros lenguajes más modernos se impongan "por la fuerza", esto es, superando en número de líneas de código al resto de lenguajes, para desarrollar en Linux es vital familiarizarse con el entorno de herramientas para C. En el presente artículo, veremos aspectos básicos en los tres componentes principales: el compilador (gcc), el depurador (gdb) y el integrador (gmake).

INSTALANDO EL ENTORNO

GNU es un grandioso proyecto que intenta producir un clónico Unix, con disponibilidad pública de las fuentes, para todas las arquitecturas más populares. Tan ambicioso proyecto tiene varios "derivados", que aprovechan parte de los programas GNU existentes, y que incorporan sus propias herramientas en el resto del sistema operativo. Tal es el caso de Linux, donde el sistema de desarrollo utilizado es el que proporciona GNU. De ahí la 'g' inicial en algunas de los comandos que tratamos con frecuencia, como gzip o gtar. Pues bien, las herramientas que trataremos en este artículo pertenecen a dicha categoría: gcc (GNU cc), gdb (GNU debugger) y gmake (GNU make).

Si usted ya se ha familiarizado con la distribución Slackware, podrá imaginar que todos estos componentes se encuentran en la serie D (desarrollo). Lo que deberemos instalar antes de seguir adelante, de la distribución 2.2, es:

d1/gcc263	El compilador de C
d2/include	Ficheros de declaraciones y cabeceras C
d3/extralib	Librerías de ayuda a la depuración
d3/gdb	El depurador
d5/libc	Las librerías de C
d6/lx123_2	Cabeceras de C adicionales
d8/binutils	Utilidades auxiliares
d8/gmake	El integrador

Antes de continuar, se debe decir que estamos suponiendo un mínimo conocimiento del lenguaje C. Si no es así, podrá hacerse una idea de las posibilidades del entorno, pero no podrá aprovechar prácticamente nada de lo que se expondrá aquí.

Se incluye en el disco un programa ejemplo (dividido en tres ficheros fuente) que realiza una conversión trivial en hexadecimal, y cuya única utilidad será practicar con los comandos de ejemplo que veremos a continuación.

EL COMPILADOR

A diferencia de entornos de desarrollo más elaborados del mundo DOS/Windows, este compilador de C se basa en línea de comando, siendo por ello muy versátil, aunque poco atractivo para el programador novel. Frente a la desventaja de su crudeza, encontramos multitud de facetas agradables. Añadiendo extensiones a la instalación básica, es capaz de producir ejecutables para otros sistemas, como MSDOS, y otras arquitecturas, como el MC68000 (Esta capacidad se denomina compilación cruzada). Dichas extensiones no se encuentran en la distribución estándar, y las comentamos aquí para que el lector conozca su existencia y pueda evaluar las posibilidades de desarrollo en Linux.

Como podemos ver en la figura 1, "gcc" es un compilador de varios pasos.

En primer lugar, preprocesa las directivas y macros, tal y como hace el programa "cpp". En un segundo paso, traduce las instrucciones C a ensamblador, optimizando el código si así se indica. Posteriormente ensambla este resultado, y enlaza el código objeto que resulta con otros posibles módulos y librerías, produciendo el ejecutable resultante. La forma mas sencilla de invocarlo es:

cc miprog.c

Esto compila el fuente indicado, y produce el ejecutable de salida, con el nombre por defecto "a.out". Para evitar este nombre tan poco explícito, utilizaremos el parámetro "-o", así:

cc -o miexe miprog.c

Lo que produce como resultado un ejecutable llamado "miexe". Además, podemos compilar tantos ficheros fuente simultáneos como deseemos. Por ejemplo:

cc -o miexe hex2int.c int2hex.c main.c

El compilador admite como entrada, no sólo ficheros fuente en C, sino también otros tipos, a los que distingue mediante la extensión del nombre referenciado. Así, se admiten ficheros ya preprocesados (.i), ensamblador (.s), objetos (.o) y librerías (.a). Se les incorpora en la fase adecuada de la compilación, tal y como veíamos en la figura 1. Por ejemplo:

cc -o miexe hex2int.c int2hex.s milib.a

Esto compilaría hex2int.c, ensamblaría int2hex.s, y enlazaría a ambos con la librería milib.a (además de librerías por defecto, como libc.a), produciendo un único ejecutable resultante.

Por otra parte, podemos parar la compilación en una fase concreta. La opción "-o" nos servirá, independientemente del tipo de resultado, el nombre del fichero de salida. Las posibilidades son:

cc -o hex2int.i -E hex2int.c
cc -o hex2int.s -S hex2int.c
cc -o hex2int.o -c hex2int.c

TABLA 1

b)reak	c)ontinue	d)elete
f)rame	h)elp	i)nfo
j)ump	k)ill	l)ist
n)ext	p)rint	r)un
s)tep	t)hread	u)ntil

Abreviaturas de comandos gdb.

La primera línea servirá para el preprocesado de código fuente. La segunda produce un fuente de ensamblador, mientras que la tercera produce un objeto sin enlazar. El uso de estas posibilidades es diversa, pero principalmente se utiliza, en primer lugar, para saber el código ensamblador que se genera cuando el compilador optimiza código, y para realizar procesado adicional con el integrador "make" como veremos en otro apartado.

Por cierto, la optimización de código se realiza mediante la opción "-O":

cc -S -O0 -o sam0.s hex2int.c
cc -S -O1 -o sam1.s hex2int.c
cc -S -O2 -o sam2.s hex2int.c

Observe el código resultante en todos los casos. El nivel de optimización '0' es el menor, e indica que el compilador no debe realizar ninguna en absoluto. El nivel '1' optimiza el uso de registros en variables locales. Los niveles posteriores realizan optimizaciones más complejas, aunque retardan el tiempo de compilación y aumentan los recursos necesarios para realizarla.

La última opción interesante a nivel básico es "-g", que indica al compilador que incluya información de depuración junto con el ejecutable, para utilizar el programa "gdb", como veremos a continuación. Dicha información sólo se puede proporcionar en el paso de compilación. En otras palabras, esta opción no produce información simbólica en ficheros de ensamblador u objeto que no hayan sido generados previamente con dicha opción.

El entorno de desarrollo en Linux se basa en la línea de comandos

EL DEPURADOR

Debido a la herencia multiplataforma del proyecto GNU, esta herramienta dispone de características "de lujo" que quizá el usuario medio nunca necesite. Entre ellas, podemos encontrar las siguientes:

- Depuración de diferentes fuentes (C, C++ y Modula-2).
- Depuración de diferentes arquitecturas (x86, 68k, MIPS, SPARC...)
- Depuración post-mortem
- Depuración en tiempo real.
- Depuración remota, por línea serie o TCP/IP.

Por enumerar las más significativas. Sin embargo, para nuestro uso, será suficiente con las prestaciones clásicas, al estilo de los conocidos programas de depuración del mundo DOS.

De nuevo, esta herramienta es muy funcional, pero poco vistosa, y puede intimidar algo en un primer contacto. Sin embargo, en este caso se dispone de un recubrimiento para X-Windows, denominado "xgdb", que resulta más amigable. Instálela si lo desea de la serie XAP.

Siguiendo el ejemplo que hemos incluido en el disco, y que se habrá compilado con:
cc -g -o miexe int2hex.c hex2int.c main.c

Se carga el depurador con:

gdb miexe

Tras algunos indicativos, se nos muestra la línea de comando, mediante la que deberemos introducir todas las órdenes. Como ayuda, tendremos las mismas posibilidades que con la shell estándar (bash), es decir, tabulador para completar palabras, y cursores para histórico de comandos y edición. Adicionalmente, podemos emplear iniciales que abrevien los comandos más frecuentes, como se observa en la tabla 1.

En primer lugar, podemos examinar el código, tanto fuente (list) como

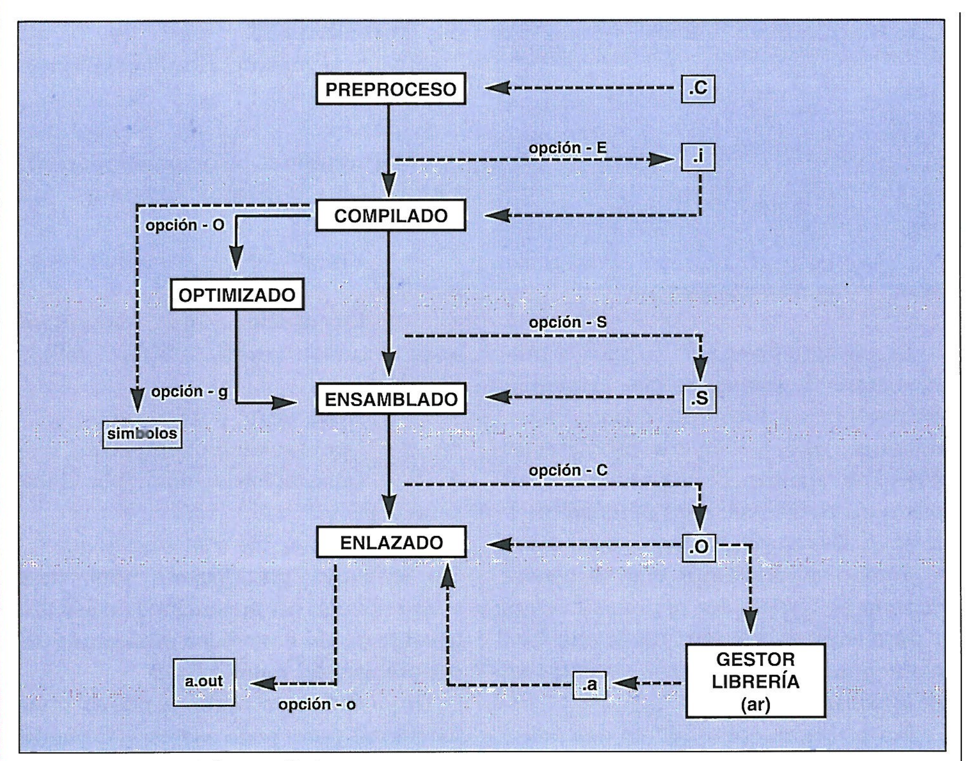


Figura 1. Etapas del compilador.

desensamblado (disassemble). Teclee los siguientes comandos, y observe los resultados que se producen.

list main.c
list main.c:1
list 1
list
list hex2int.c:1
list 1
l int2hex
disassemble int2hex

El primero produce un resultado erróneo, porque indicamos un fichero fuente, pero no a partir de qué línea mostrarlo.

Sin embargo, el segundo ejemplo muestra un cierto trozo de código, a partir de la primera del fichero 'main.c'.

El tercer comando nos demuestra que también podemos indicar el número de línea a secas.

El cuarto nos muestra un error. Éste se ha producido porque el comando muestra líneas consecutivas. Es decir, suponiendo que la pantalla muestra 24 líneas, y que tenemos 70 en nuestro código, el primer comando 'list' mostrará las primeras 24 líneas, el segundo las siguientes, el tercero mostrará 22, y el cuarto producirá un error. Como ya habíamos mostrado el fichero completo con el anterior comando, en esta ocasión se nos indica el fin del fichero.

El quinto ejemplo nos muestra todas las líneas de otro fichero fuente, desde la primera.

El sexto indica que ha cambiado el fichero "por defecto". Como podemos suponer, cada vez que pidamos un fichero fuente de forma explícita, operaremos con él hasta que se indique otra cosa.

El penúltimo comando nos indica que podemos listar funciones completas. De paso puede observar que podemos abreviar el comando "list" mediante su primera letra.

Y el último nos enseña la forma más básica de desensamblar una función.

Tras dar un breve repaso a la forma de mostrar código, veamos el manejo de los breakpoints mediante los siguientes comandos:

break main.c:2
info break
info b
d b 1
i b

En la primera linea se establece un punto de ruptura en la segunda línea del fichero "main.c".

La segunda y la tercera enseñan los breakpoints activos. Observe que tienen un número a la izquierda que los identifica de forma única. Fíjese que también podemos abreviar la palabra "break".

En la tercera aparece un nuevo comando (delete), abreviado. La línea equivale a delete break 1, y borra el punto numerado como "1". Recuerde que dicha numeración aparece al mostrar todos los puntos activos.

Finalmente, la última línea muestra, abreviadamente, los puntos activos. El comando completo rezaría info break.

Prestemos ahora atención a la forma de controlar la ejecución de nuestro programa. Teclee los siguientes comandos de ejemplo y observe lo que ocurre:

d
step
run
b main.c:1
r
s
s 3

En primer lugar, indicamos "delete" de forma abreviada. Esto borra todos los puntos de ruptura de nuestro programa.

A continuación, hemos intentado ejecutar paso a paso nuestro programa. Como todavía no se encuentra en ejecución, se nos indica con un error.

La tercera línea manda ejecutar nuestro programa. Dado que no indicamos nada especial, la ejecución se realiza de un tirón, hasta que el programa finaliza. Esto no nos resulta útil en absoluto.

De modo que introducimos un punto de ruptura en la primera línea del fichero principal, y ejecutamos de nuevo nuestro programa (observe que el comando "run" se ha abreviado). Esta vez se indica que se ha alcanzado la línea de breakpoint pedida, y el programa se detiene. Ahora sí podemos ejecutar paso a paso.

La línea "s 3" indica que queremos ejecutar exactamente tres pasos. El cumplimiento de este comando produce un error. Esto se debe a que pasamos a ejecutar líneas de código de las que no tenemos fichero fuente disponi-

ble (en concreto, nos encontramos dentro del código de la función sprintf. A menos que tenga los fuentes de las librerías instaladas, el depurador no puede mostrar ninguna información al respecto). Sigamos introduciendo comandos:

bt
u int2hex.c:5
display /x i
s
display i
s
c

Con el comando "bt" indicamos "backtrace", y queremos decir al depurador que muestre el estado de anidamientos en la ejecución del programa. Podemos observar así el estado actual de la pila, con todas las llamadas a funciones, los parámetros pasados, y la línea y fichero en que se han producido.

La siguiente línea es algo más complicada, y muestra un nuevo comando (until). Indicamos aquí que la ejecución prosiga hasta encontrar la línea 5 del fichero *"int2hex.c"*. Si observamos el estado de anidamientos (bt), es justo la línea que sigue a la llamada a sprintf. De algún modo, es como si hubiésemos puesto un punto de ruptura en esa línea, hubiéramos ejecutado el programa, y al parar hubiésemos borrado el breakpoint.

El siguiente comando es una muestra de visualización de variables. Indica que queremos visualizar la variable denominada "i" en hexadecimal. Puede observar los efectos con el siguiente comando (step). Cada vez que pare tras un punto de ruptura, o ejecute paso a paso, o con un until, se visualizará la variable en el formato indicado.

Igualmente puede ver las variables en decimal, con el siguiente comando del ejemplo. Fíjese que, ahora, la ejecución de cada paso de programa muestra las dos vistas pedidas, en hexadecimal, y en decimal.

Finalmente, la última línea (continue), reanuda la ejecución hasta el próximo punto de ruptura, o hasta el fin del programa, que es nuestro caso.

CUADRO 1

```
todo:          miexe

hex2int.o:     hex2int.c
               cc -c hex2int.c

int2hex.o:     int2hex.c
               cc -c int2hex.c

miexe:         hex2int.o int2hex.o main.c
               cc -o miexe hex2int.o int2hex.o
main.c

debug:         hex2int.o int2hex.o main.c
               cc -g -o miexe hex2int.o int2hex.o
main.c
               gdb miexe
               rm miexe

clean:
               rm *.o miexe
```

El fichero de reglas: makefile.

Como puede observar, en todo momento el depurador considera como "paso" de programa una línea de fuente. El total de líneas disponibles se puede comprobar con el comando "list" ya visto.

Hemos avanzado algo de la forma en la que podemos ver los datos. El comando "display" es lo que se denomina watchpoint, o "espía". Indicamos una variable de la que queremos saber su valor de forma continua, mientras el programa ejecute.

El comando print tiene el mismo formato, pero se utiliza cuando deseamos conocer un valor de forma puntual. Una vez mostrado el resultado, no hay efecto posterior.

Disponemos de un recubrimiento para X-Windows del depurador: xgdb

El comando x, finalmente, muestra datos de memoria, en forma de volcado.

También podemos encontrar cierto paralelismo entre la forma de manejar breakpoints y watchpoints.

En ambos casos, la forma de crear, consultar y eliminarlos, respectivamente, son:

<comando> <ítem>
info <comando>
delete <comando> <núm_ítem>

Finalmente, con quit salimos del depurador. Si tuviéramos un programa a mitad de ejecución, se nos pediría confirmación antes de acabar la sesión.

Con esto, damos por terminado este breve recorrido por las opciones de depuración. Por supuesto, hemos dado una visión simple de las amplias posibilidades de esta herramienta, pero esperamos que suficiente para que tome una visión general.

EL INTEGRADOR

Para pequeños programas, con quizá un sólo fichero fuente, resulta sencillo el proceso de editar, compilar y depurar. Imagine, por el contrario, un proyecto de cierta envergadura, con varios ficheros relacionados entre sí, donde se necesita seguir un cierto orden a la hora de ser construidos. Repetir varias veces el proceso de compilación puede ser, como mínimo, tedioso.

Se podría pensar en generar un fichero de comandos de shell que agrupe todos los pasos necesarios. Sin embargo, esta aproximación tiene el inconveniente de que la modificación de una sola línea de un sólo fuente produciría la recompilación completa de todo el proyecto. Como podemos suponer, esto vuelve a ser poco práctico.

Si tuviéramos una forma de indicar, mediante reglas, los ficheros fuente necesarios para construir un objeto concreto, y los objetos y librerías necesarios para que su enlace produzca nuestro programa, alguna herramienta podría analizar dichas reglas para encontrar el mínimo de operaciones necesarias con las que obtener nuestro ejecutable. Pues bien, este programa existe, y se denomina "make". Las reglas que entiende tienen la forma:

```
resultados: fuentes
        comandos
```

Todas las separaciones requieren tabuladores, y la forma de distribuir las líneas es obligatoria. Se indica que se debe comprobar la lista de "fuentes" para saber si debemos reconstruir la lista de "resultados". Si fuese así, la herramienta ejecutará para ello los "comandos".

La decisión, sobre si hay que ejecutar dichas acciones se basa en las

fechas de los ficheros. Es decir, si las fuentes son más antiguas que los resultados, no se toma ninguna acción. Sin embargo, si las fuentes son más recientes, se interpreta que se debe rehacer el resultado, dado que se ha producido alguna modificación.

Como siempre, lo mejor es verlo en un ejemplo. Observe el cuadro 1. Este fichero se denomina makefile. Cuando invocamos el comando:

make

desde el directorio donde se encuentra este fichero de reglas, esta herramienta busca la primera del fichero, y mira en la lista de fuentes de dicha regla (en este caso, miexe). Una vez hecho, comprueba si hay reglas que tengan a dicho fuente como objetivo (la cuarta, en el ejemplo). De nuevo se evalúa la lista de fuentes, buscando nuevas reglas que indiquen como obtenerlos. Encontramos la segunda y tercera regla para "hex2int.o" e "int2hex.o" respectivamente, pero no hay ninguna para "main.c". En este último caso, se le supone un fuente proporcionado por el usuario, y se le busca en el disco. Si no existe, se aborta todo el proceso, y la utilidad indica "Don't know how to make main.c". Es decir, no tiene disponible un fichero fuente, ni tiene reglas que le indiquen cómo se genera. Si el fichero existe, se siguen evaluando todas las reglas, hasta llegar a las que sólo tienen ficheros originales, que deban encontrarse en el disco.

Ahora se comparan las fechas de estos ficheros con las de los objetivos de la izquierda de las reglas en las que son fuentes. Si el objetivo no está en el disco, o su fecha es más antigua, se ejecutan los comandos asociados. De esta forma, se pueden llegar a ejecutar nuevas reglas y comandos, que finalmente desencadenen una recompilación del proyecto, o bien puede evaluarse todo sin que sea necesario realizar ninguna operación. Todo depende de la relación de fechas entre los ficheros.

Si ha entendido la explicación anterior, observará que hay reglas que nunca se evalúan en el proceso. En nuestro ejemplo, las dos últimas. Suelen indicar operaciones especiales, y para usarlas se deben indicar parámetros al invocar la herramienta. En el ejemplo:

make clean
make debug

El primer comando limpia los ficheros intermedios de la compilación. Se suelen emplear reglas de este estilo para que la siguiente ejecución de make recompile todos los ficheros desde el principio, o para ahorrar espacio cuando ya se ha obtenido el resultado deseado.

El segundo se parece a la regla de miexe, pero produce un ejecutable con el que se arranca automáticamente al depurador. Esto es un ejemplo de la multitud de comandos que se pueden ejecutar de esta forma.

Observe que estas reglas nunca producen los ficheros objetivo correspondientes (que en buena lógica se denominarían debug o clean). En realidad, los nombres de la izquierda de este tipo de reglas tienen un sentido ilustrativo. Cada vez que la herramienta pasa por ellas, los comandos asociados son ejecutados incondicionalmente, dado que no debe haber ficheros en el disco cuyo nombre coincida, y que nunca se producirán como consecuencia de la ejecución de los comandos correspondientes.

Por último, fíjese que en la primera regla no hay comandos asociados. Ésta representa una forma de construir sinónimos ("make todo" equivale a "make miexe", y, en este caso, como es la primera regla del fichero, equivale a "make"). Si recordamos que cuando ejecutamos la herramienta sin parámetros se evalúa la primera regla del fichero makefile, al ser expresada así podemos independizar el orden de dichas reglas, lo cual lo hace más legible. En este caso, el comando sin parámetros revisa la construcción del ejecutable miexe.

Como en los casos anteriores, hemos visto aspectos muy básicos de esta utilidad, sin profundizar apenas. Esta herramienta en particular tiene la virtud de poder expresar las mismas reglas con ficheros makefile más pequeños pero mucho más complejos, inadecuados para una primera toma de contacto.

El compilador ANSI 'gcc' puede generar código cruzado o nativo

'make' automatiza la reconstrucción de proyectos complejos

CONCLUSIONES

Se ha pretendido en este artículo guiar al lector sobre la forma general de compilar aplicaciones en Linux, sin aburrir con explicaciones demasiado detalladas. Aquellas personas que deseen profundizar en estos temas, pueden acudir a manuales de referencia de cualquier Unix, al man (detallado en el caso del compilador, pero algo escaso con el resto de herramientas), y a los manuales GNU de las tres utilidades. Incluimos con la revista el fichero *PostScript* con el manual del depurador. Los afortunados poseedores de una impresora láser que entienda este formato tienen la documentación más detallada disponible. El resto, deberán conformarse con utilizar ghostview, un visualizador de este lenguaje para el PC. Finalmente, y para usuarios más lanzados, puede ser de utilidad observar ejemplos de programas grandes de los que se disponga de los fuentes. Cualquier paquete para Linux, o el mismo núcleo, puede servir para aprender las posibilidades mas complejas.

En definitiva, esperamos haber ayudado una vez más a ampliar sus conocimientos sobre Linux. Hasta otro mes.

CONCEPTOS BÁSICOS (3ª PARTE): BASES DE DATOS

Juan Manuel y Luis Martín

Como continuación del artículo anterior de la serie, en el que analizábamos algunos aspectos de manejo de bases de datos en Clipper, en el presente artículo se realiza una descripción más profunda de dos aspectos que, por su complejidad, necesitan una explicación más detallada y extensa.

Como ya se explicó en el artículo anterior, las bases de datos manejadas por Clipper son ficheros que almacenan información relativa a entidades del mundo real con un formato especial, similar a una tabla. Cada uno de los registros (filas) contiene información relativa a una entidad, en forma de propiedades o atributos que se almacenan en los distintos campos (columnas). El número de registros es variable, mientras que el de campos es siempre fijo.

Los tipos de campos que pueden contener las bases de datos se corresponden con los tipos de datos que maneja Clipper y que, en general, son similares a los de otros lenguajes de programación. Sin embargo, existe un tipo de campo especial que sólo existe en los lenguajes de programación orientados al manejo de bases de datos. Se trata de los campos memo, cuyo tratamiento y formato de almacenamiento difiere bastante del resto, por lo que debe ser analizado con mayor detalle.

Las operaciones que pueden realizarse sobre los diferentes campos de los registros de una base de datos, pueden estar basadas tanto en la posición del registro como en el contenido de sus campos. Por ello, la mayoría de los mandatos de tratamiento de registros llevan asociadas unas cláusulas que permiten delimitar sobre qué registros tendrá efecto la operación correspondiente. El dominio de estas cláusulas es fundamental para la correcta utilización de los mandatos, por lo que también deben ser estudiadas con mayor profundidad.

LOS CAMPOS MEMO

Los campos memo constituyen un tipo de datos especial para el almacenamiento de textos de longitud variable en las bases de datos. Se trata de campos capaces de almacenar cadenas de hasta 65.535 caracteres. Su funcionamiento es similar al de los campos de tipo carácter, aunque existen algunas diferencias importantes en cuanto a su almacenamiento.

Los campos de tipo memo se almacenan en un fichero independiente de la base de datos, que tiene el mismo nombre que ésta, pero la extensión DBT. Dentro del archivo DBF que constituye la base de datos, sólo se almacena un pequeño campo de 10 bytes que contiene un puntero. Este puntero indica la posición que ocupa el dato dentro del archivo DBT. El contenido de los archivos destinados a almacenar campos memo está estructurado en bloques de 512 bytes. En el campo de fichero DBF figura el número del bloque en el que comienzan los caracteres correspondientes del fichero DBT.

Los campos de tipo memo se almacenan en un fichero independiente de la base de datos

TABLA 1	
Función	**Descripción**
HARDCR()	Sustituye todos los retornos de carro automáticos por forzados
MEMOEDIT()	Muestra o edita cadenas de caracteres y campos memo
MEMOLINE()	Devuelve una línea de texto de una cadena de caracteres o campo memo
MEMOREAD()	Devuelve el contenido de un fichero de disco como cadena de caracteres
MEMOTRAN()	Sustituye un retorno de carro/salto de línea en cad enas de caracteres
MEMOWRIT()	Escribe una cadena de caracteres o campo memo en un fichero de disco
MLCOUNT()	Devuelve el número de líneas de una cadena de caracteres o campo memo
MLCTOPOS()	Devuelve la posición en bytes basada en la posición de fila y columna
MLPOS()	Determina la posición de una línea en un campo memo
MPOSTOLC()	Devuelve la posición de fila y columna basada en la posición en bytes
READINSERT()	Activa/desactiva el modo de inserción actual para Memoedit()
SET SCOREBOARD	Activa/desactiva la visualización de mensajes desde Memoedit()

Funciones específicas para el tratamiento de campos memo.

Los operadores que pueden aplicarse a los campos memo son, en general, los mismos que se aplican a los datos de tipo carácter. Sin embargo, es necesario tener en cuenta que no existe una representación literal para datos de tipo memo, ya que éstos sólo pueden existir como campos de una base de datos, y no como operandos con representación en memoria.

Para el manejo de campos memo, Clipper dispone de un conjunto de funciones propias, además de las de tratamiento de cadenas, que se muestran en la tabla 1. Como puede verse, la mayoría de ellas están relacionadas con la edición de textos, siendo la función MemoEdit() la más importante. Se trata de una función que permite realizar la edición de textos de cadenas de caracteres, y campos memo en un recuadro o ventana de la pantalla, haciendo que éste se comporte como un pequeño editor de textos. Para ello, se utiliza una zona de memoria intermedia a modo de buffer. La sintaxis de esta función es la siguiente:

MEMOEDIT(cTexto, nSup, nIzq, nInf, nDer, lModo, cFunción, nLongLínea, nTab, nFilaMemo, nColMemo, nFilaVent, nColVent)

Como puede apreciarse, consta de multitud de argumentos, todos ellos opcionales, que permiten personalizar su funcionamiento. En el argumento cTexto se indica la cadena de caracteres o el campo memo que se desea editar. Si se omite este argumento, Clipper utiliza un buffer vacío. Esta función devuelve el texto editado en forma de cadena de caracteres. Si lo que se edita es un campo memo, los cambios quedarán reflejados en el disco.

Los argumentos nSup, nIzq, nInf y nDer indican las coordenadas de la ventana de visualización, que por defecto será toda la pantalla. El argumento lModo indica si se permite la edición o, por el contrario, sólo es posible visualizar el texto.

```
// Carga la variable cTexto
// desde una ventana de edición
cTexto := MEMOEDIT(,10,10,40,15,.T.)

// Muestra el campo memo OBSERVAC
// en una ventana de visualización
MEMOEDIT(FIELD->observac,10,10,40,15,.F.)
```

Es importante tener en cuenta que cuando se ha definido un cierto tamaño para la ventana de edición, la longitud del texto puede ocupar más de una línea. En estos casos la función va insertando retornos de carro automáticos en el texto, con el fin de situar éste en el espacio de pantalla disponible. Para ello, se utilizan los caracteres CHR(141) y CHR(10). Sin embargo, el usuario también puede insertar retornos de carro manuales, pulsando la tecla <Intro>. En este caso, los caracteres insertados serán CHR(13) y CHR(10). Una vez finalizado el proceso de edición, pueden utilizarse las funciones *HARDCR() Y STRTRAN()* para cambiar los retornos de carro automáticos por manuales, o simplemente para eliminarlos.

Como ocurre con otras funciones, es posible indicar el nombre de una función de usuario en el argumento cFuncion, para personalizar su funcionamiento interno. El resto de argumentos permiten configurar el aspecto de la presentación, y pueden consultarse en la ayuda on-line ofrecida por las Guías Norton.

Para rellenar un campo memo (o una variables de tipo carácter) con el contenido de un archivo de texto individual, puede utilizarse la función *MEMOREAD()*. Esta función devuelve el contenido del archivo de texto especificado, en forma de cadena de caracteres. El tamaño máximo del archivo está limitado a 64 Kb.

```
// Rellena el campo memo TEXTO
// con el fichero DATOS.TXT
FIELD->texto := MEMOREAD( "datos.txt" )
```

Igualmente, para almacenar el contenido de un campo memo (o de una cadena de caracteres) en un fichero individual, puede utilizarse la función MEMOWRIT(). Esta función devuelve .T. cuando la operación se ha realizado satisfactoriamente, y .F. en caso contrario.

```
// Salva en el fichero OBSERV.TXT
// el contenido del campo memo OBSERV
MEMOWRIT( "observ.txt", FIELD->observ )
```

LA CLÁUSULA DE ÁMBITO

La mayoría de los mandatos que realizan operaciones masivas con los registros de una base datos permiten especificar el ámbito de actuación, así como condiciones lógicas de tipo *FOR* y *WHILE* que deben cumplir los registros. La cláusula de ámbito permite delimitar sobre qué registros se debe realizar la operación, en función de la posición que éstos ocupan en la base de datos. De esta forma, la operación queda confinada a una determinada zona de la base de datos. La tabla 2 muestra los mandatos de tratamiento de registros que permiten la utilización de las cláusulas ámbito, *FOR* y *WHILE*.

Los valores posibles de esta cláusula pueden expresarse mediante cuatro tipos de sintaxis distintas:

- ALL: Indica que la operación se realizará sobre todos los registros de la base de datos. Cuando se omite la cláusula, éste es el valor que se asume por defecto.

- REST: Indica que la operación se realizará sobre los registros situados desde el actual (inclusive), hasta el final de la base de datos.

- NEXT n: Indica que la operación se realizará sobre un cierto número de registros situados a partir del actual (inclusive). El número de registros se especifica en el argumento n.

- RECORD n: Indica que la operación se realizará sólo en el registro indicado en el argumento n.

El orden en que van siendo procesados los registros depende del índice que se encuentre activo, salvo para el caso de la cláusula RECORD, ya que ésta limita su actuación a un determinado registro. Las siguientes sentencias ilustran el funcionamiento de estas cláusulas:

// Pone a 0 el campo valor de todos los registros
REPLACE ALL valor WITH 0
// Cuenta el número de registros a partir del actual
COUNT TO num REST
// Desmarca los 10 siguientes registros
RECALL NEXT 10
// Borra el registro 100
DELETE RECORD 100

Si se omite la cláusula de ámbito, el valor por defecto es *ALL*, salvo en el caso de los mandatos *DELETE, RECALL y REPLACE.* En estos mandatos, cuando no ha sido especificada una cláusula *FOR* o *WHILE*, el valor por defecto del ámbito es *NEXT 1*, es decir, el registro actual.

// Marca el registro actual
DELETE
// Pone a 0 el campo valor de los registros marcados
REPLACE valor WITH 0 FOR DELETED()

LAS CLÁUSULAS FOR Y WHILE

Las cláusulas FOR y WHILE permiten especificar una condición lógica que deben cumplir los registros sobre los que actuará el mandato, generalmente en función del contenido de sus campos. Estas cláusulas sólo se evalúan sobre los registros especificados en la cláusula de ámbito. La diferencia entre ambas consiste en que la cláusula *WHILE* termina la ejecución del mandato la primera vez que falla su condición lógica, mientras que la cláusula *FOR* continúa la ejecución. La mejor forma de comprender las diferencias de funcionamiento entre ambas es con un ejemplo sencillo:

DELETE WHILE valor = 0

En este caso, si el campo valor del primer registro de la base de datos no ésta a 0, el mandato sólo se ejecuta para dicho registro, cancelándose la ejecución. Como consecuencia, no se marcará ningún registro. Por el contrario, si el campo está a 0, el registro es marcado y la ejecución del mandato continúa con el siguiente registro, y así sucesivamente.

DELETE FOR valor = 0

En este otro caso, el mandato recorrerá todos los registros de la base de datos, independientemente del contenido de los registros. Por tanto, serán marcados todos aquellos registros cuyo campo valor esté a 0.

La utilidad de la cláusula *WHILE* se justifica en el caso de bases de datos indexadas, ya que permite recorrer zonas restringidas de la base de datos en función del contenido de sus campos, con el consiguiente ahorro de tiempo de proceso. Así, por ejemplo, si la una base de datos de está indexada por el campo valor, para borrar aquellos registros cuyo campo valor esté a 100, puede optarse por dos opciones:

// Recorriendo toda la base de datos
DELETE FOR valor = 100
// Recorriendo solo los registros necesarios
SEEK 100
DELETE WHILE valor = 100

La utilización combinada de las cláusulas de ámbito, *FOR* y *WHILE*, permite realizar operaciones sobre las bases de datos limitadas a conjuntos de registros con especificaciones complejas.

TABLA 2

Mandato	Descripción
AVERAGE	Calcula la media aritmética de una lista de expresiones numéricas en los registros indicados
COPY TO	Copia a un fichero los registros indicados
COUNT	Cuenta el número de registros indicados
DELETE	Marca para borrar los registros indicados
INDEX	Crea un archivo de índice para ordenar la base de datos
LOCATE	Busca el primer registro que cumple una condición indicada
RECALL	Desmarca los registros indicados
REPLACE	Sustituye el contenido de los campos para los registros indicados
SORT	Copia físicamente ordenados los registros especificados
SUM	Suma los valores de una lista de expresiones numéricas para los registros especificados
TOTAL	Resume y acumula los registros indicados por un valor clave

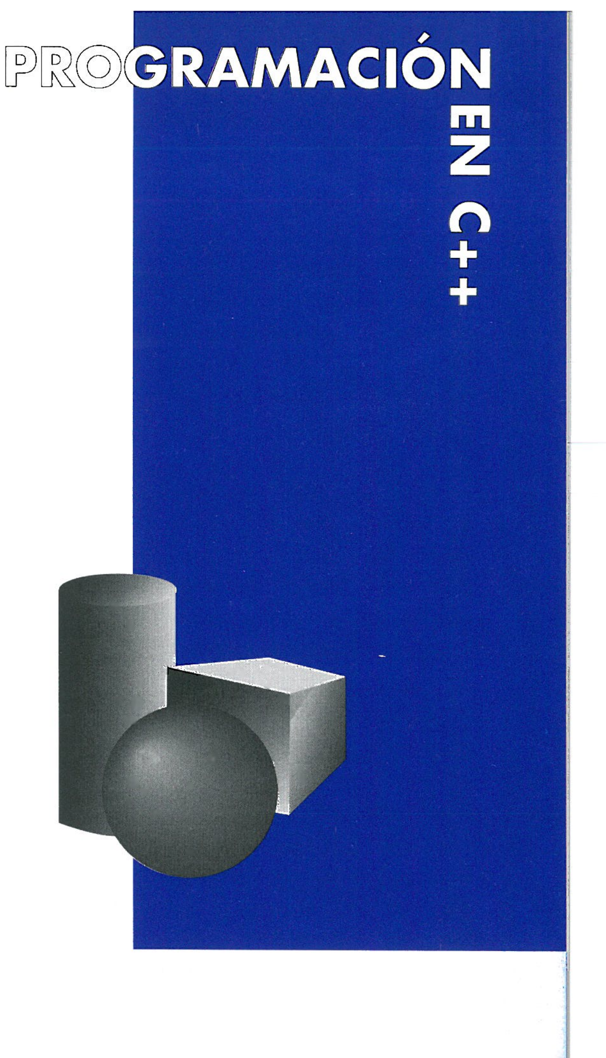
Mandatos con las cláusulas ámbito, FOR y WHILE.

CONCLUSIÓN

La mejor forma de dominar la utilización de los campos memo y de las cláusulas de ámbito, *FOR* y *WHILE*, es la realización de múltiples ejemplos sencillos a modo de prácticas. Una vez familiarizado con los conceptos básicos del lenguaje y de las bases de datos, el lector estará en condiciones de aprender a manejar los *RDD*. Se trata de librerías que permiten utilizar bases de datos en formatos distintos a los de Clipper, y que serán analizados en el próximo artículo.

SISTEMAS DE VENTANAS EN C++

Ignacio Cea



Durante estos últimos cuatro meses se han estado exponiendo los principios básicos que rigen la construcción de un sistema de ventanas. En este último capítulo, además de completar los conceptos, se desarrollará un ejemplo completo que usted podrá adaptar para desarrollar sus propios sistemas de ventanas, además de poder explorar en un interesante campo.

Los conceptos necesarios para el desarrollo del sistema de ventanas ya han sido descritos. Falta, sin embargo, dar al lector un ejemplo completo y práctico con el que no sólo poder comprender mejor las anteriores entregas sino con el que poder explorar nuevas ideas o, incluso, poder emplearlo dentro de sus propias aplicaciones. Esta es precisamente la meta del artículo que hoy tiene entre sus manos. Ya que el sistema a presentar está desarrollado íntegramente dentro de un modo gráfico VGA de alta resolución, no habrá más remedio que hablar, antes de nada, de otras clases que nos faciliten el dibujo en el mismo.

UN GESTOR GRÁFICO

Son muchos los lugares dentro del sistema de ventanas desarrollado donde será necesario el tratamiento de gráficos. En principio no se van a emplear otras órdenes que no sean las que el compilador suministra por defecto dentro de su librería "graphics.h" (*setviewport*, *line*, *circle*, etc..).

Sin embargo, no es de extrañar que los lectores más avezados quieran que el sistema funcione de manera idéntica dentro de modos gráficos no soportados por las BGI del compilador de Borland, y para los que ellos habrán desarrollado sus propias rutinas.

Si, teniendo en cuenta esto, el empleo de los gráficos no se centralizara, por ejemplo, en alguna clase, se obligaría al usuario a realizar una tediosa labor de búsqueda y sustitución de todas las llamadas a funciones de dibujo cada vez que quisiera emplearlo para otro modo que no fuera el inicialmente pensado. ¿Qué sería entonces del tan mencionado reuso?.

Se necesita, por tanto, alguien neutral (una clase) que se encargue de realizar todas las labores de dibujo.

Se necesita alguien neutral (una clase) que se encargue de realizar todas las labores de dibujo

LA CLASE PANTALLA

El pensar en una clase para tal menester hace que, si se deseara emplear el entorno con cualquier otro sistema gráfico, disponible o no dentro del compilador, fuera la implementación de sus métodos internos el único sitio donde hubiera que tocar código.

Sería algo similar a, dentro de una empresa, cambiar un empleado por otro, en lugar de alterar la forma de trabajar de todos los demás. Es algo, sin duda más "barato", pero con lo que, evidentemente, se pierde rendimiento y velocidad de ejecución. Sin embargo, hay que recordar que el rendimiento no es uno de los parámetros fundamentales dentro de la orientación a objetos y sí, por el contrario, el reuso. Por otro lado, las máquinas actuales tienen la suficiente rapidez y capacidad como para ni enterarse de ese cambio. El nombre de la nueva clase será "Pantalla".

LA IMPLEMENTACIÓN DE LA CLASE

Jamás se ha hablado, por otro lado, de que el entorno desarrollado tuviera la

posibilidad de trabajar con varias ventanas simultáneamente (como si lo hace, por ejemplo, Motiff de UNIX), por lo que dentro de cualquier aplicación que considere, sólo existirá un objeto de la clase "Pantalla".

Sin duda, que la acción del programador podría hacer fácilmente que existieran varios de estos objetos (basta, simplemente, con inicializarlos), pero aún así hay que asegurar que las variables internas modificadas de la clase fueran, sea cual fuere el objeto invocado, siempre las mismas. Esto hace pensar en una clase en la que todos sus atributos fueran estáticos y también, por tanto, los métodos que los manejan.

Como ya se indicó en artículos anteriores, la forma de pintar una ventana era: dividirla en todas los distintos rectángulos elementales que la forman, restringir la salida a pantalla dentro de cada uno de ellos y, por último, realizar el dibujo. Los atributos que hay dentro de la clase "Pantalla" guardan, de un lado, las coordenadas de la zona de pantalla donde actualmente esté restringida la salida y, de otro, la posición de la misma considerada como origen de coordenadas.

Los métodos, por otro lado, fijan esa zona restringida, cambian el origen de coordenadas y pintan distintas figuras sobre la pantalla. Observe la figura 1.

LA FORMA DE DIBUJAR

Las anteriores entregas de esta serie se detuvieron más bien poco en la forma en la que las cosas habrían de ser dibujadas dentro del entorno. Tan sólo se habló de un método para dibujar figuras dentro de una ventana (clase Ventana, método dibujar y clase Objeto) cuyo algoritmo consistía en recorrer las distintas zonas elementales que la formaban, restringiendo la salida a cada una de ellas e invocando, a continuación, al método dibujar del objeto pasado como parámetro aprovechando que tal función había sido declarada como virtual. Es hora, sin embargo, de profundizar un poco más en la materia.

DIBUJANDO LA VENTANA

En un sistema de ventanas, no sólo hay que dibujar cosas dentro de la ventana sino que, además hay que dibujar la propia ventana en si. Esta, es una tarea tan importante que se repite, por ejemplo, cada vez que se mueve la ventana o cuando un nuevo objeto de esa clase es creado.

Es imprescindible, por tanto, crear un método con esas responsabilidades. Sin embargo, ya habrá caído en la cuenta de que, el dibujo de una ventana es distinto en cada una de ellas, aunque la forma de pintarlo siga el mismo algoritmo que no es otro que el ya descrito para dibujo de objetos individuales.

El método, por tanto, ha de ser dividido en dos partes: una común a todos los objetos de la clase ventana (definida en esta) que contenga el algoritmo de dibujo y otra, propia de cada ventana (virtual y redefinible en las hijas) que albergue el dibujo del interior. Al primero de los métodos se le denominará dibujar y al otro dibujar_trozo. Observe la figura 2 con la nueva definición de la clase.

El dibujo de una ventana es distinto en cada una de ellas, aunque siga el mismo algoritmo

Estos métodos han de ser invocados, como se dibujo antes, cada vez que se cree una ventana, o cuando cambie de posición o, incluso, al reescalarla.

ALGORITMO DE DIBUJO DE LA VENTANA

El algoritmo de dibujo de cosas dentro de ventanas ya descrito, se encargaba, para cada una de las zonas que la forman, de restringir la salida dentro de ella, y de, tras esto, pintar el dibujo. La tarea de dibujar la zona es, más bien, una responsabilidad de los objetos de esa clase en lugar del método dibujar de la clase "Ventana", pues son ellos y no otros quienes tienen la información suficiente para hacerlo (coordenadas, ventana a la que pertenece, etc...).

Por tanto el algoritmo empleado para dibujar una ventana se limitará a recorrer todas las zonas que la integran (si es su responsabilidad) pidiéndolas que se dibujen (zona -> dibujar) según se observa en la figura 3.

La clase Pantalla

```
La clase Pantalla
// Clase Pantalla
// Realizado por Ignacio Cea Forniés
// Copyright de Sólo Programadores

        #if !defined (__PANTALLA_HH__)
        #define __PANTALLA_HH__

        extern "C" {
          #include <graphics.h>
        };

// Características de la clase
class Pantalla {
  private :
   static int XP1,YP1,XP2,YP2; // Coordenadas
del hueco activo
   static int X0,Y0;   // Punto de origen del traza-
do

  public :
   Pantalla ();
   ~Pantalla () { closegraph (); }
     static void limitar_salida_en (int x1,int y1,int
x2,int y2);
   static void mover_origen_a (int x0,int y0);
    static void dibujar_linea (int x1,int y1,int x2,int
y2,int C = getcolor (),
          int P = SOLID_LINE);
    static void dibujar_rectangulo (int x1,int y1,int
x2,int y2,int C = getcolor (),
          int = SOLID_LINE);
    static void dibujar_caja (int x1,int y1,int x2,int
y2,int C = getcolor (),
          int = SOLID_FILL);
    static void dibujar_texto (int x,int y,char *txt,int
C = getcolor (),
          int F = DEFAULT_FONT,int S = 1);

};

// Métodos inline
inline void Pantalla::mover_origen_a (int x0,int y0)
{ X0 = x0; Y0 = y0; }

        #endif
```

Figura 1.

¿Y EN LAS ZONAS QUÉ?

En la clase zona, por otro lado, habrá que colocar ese método "dibujar" aludido. Este, sólo tendrá que, tras limitar

La clase Ventana

```
// Clase Ventana
// Realizado por Ignacio Cea Forniés
// Copyright de Sólo Programadores

        #if !defined (__VENTANA_HH__)
        #define __VENTANA_HH__

        #include "ZonWrite.hh"
        #include "Evento.hh"

        class NodoVentana;
        class Objeto;

// Características de la clase
class Ventana {

  protected :
    static NodoVentana *Inferior;
    static NodoVentana *Superior;
    static void zonas_ocultas (void);
    static void zonas_vistas (void);
    static Evento CSEvento;
    static Evento detectar_evento (void);
    void dibujar_antiguas_zonas_ocultas (void);

    ZonasEscritura ZonasTapadas;
    ZonasEscritura ZonasVistas;
    int X1,Y1,X2,Y2;

  public :
    Ventana (int x1,int y1,int x2,int y2);
    ~Ventana ();
    int x1 (void) const;
    int y1 (void) const;
    int x2 (void) const;
    int y2 (void) const;
    void mover (int ax,int ay);
    void reescalar (int ax,int ay);
    virtual void dibujar_trozo (void) const = 0;
    virtual void borrar_trozo (void) const = 0;
    void borrar (void) const;
    void dibujar (void) const;
    void dibujar (const Objeto &) const;
    virtual int gestionar_evento (void) = 0;
    static void servidor (void);
};

// Métodos inline
inline int Ventana::x1 (void) const { return (X1); }
inline int Ventana::y1 (void) const { return (Y1); }
inline int Ventana::x2 (void) const { return (X2); }
inline int Ventana::y2 (void) const { return (Y2); }

        #endif

// Fin del fichero Ventana.hh
// Versión 1.0
// 14 de Julio de 1995
```

Figura 2.

El método dibujar de la clase Ventana

```
/////////
//    Limpia el interior de la ventana
/////////
void Ventana::dibujar (void) const
{
    // Recorro las distintas zonas vistas...
    // ...borrando sobre cada una de ellas
    NodoZona *Pintar = ZonasVistas.primero ();
    while (Pintar != NULL) {
      Pintar -> dibujar ();
      Pintar = Pintar -> siguiente ();
    }
}
```

Figura 3. El método dibujar de la clase Ventana.

la salida a sus bordes, invocar al método de dibujar_trozo de la ventana propietario (figura 3).

"Ventana" (abstracción); cada una de ellas puede reservar su propia porción de memoria que no sería eliminada de

Se establece un diálogo entre las clases muy propio de los programas orientados a objeto

Se establece, de esta forma, un diálogo entre las clase "NodoZona" y "Ventana" muy propio de los programas orientados a objeto. Este continua comunicación afecta, irremediablemente, al rendimiento de la aplicación final pero, como ya se apuntó antes, este no ha de ser el parámetro básico de diseño a tener a cuenta.

En la figura 4 se incluye un gráfico para aclarar esta conversación, mientras que la figura 5 muestra la apariencia definitiva de la clase "NodoZona".

LA DESTRUCCIÓN DE LAS VENTANAS

Un capítulo importante que, hasta el momento, ha sido dejado en el tintero es el conjunto de pasos a seguir cuando una ventana se destruye; es decir, la implementación del destructor de la clase Ventana.

Sus misiones no son más que eliminarse de la lista de ventanas del entorno y reconstruir las zonas vistas y ocultas de aquél, tal y como muestra la figura 6.

Habrá observado que el destructor es declarado como virtual. Esto es debido que, si bien todas las ventanas (sean hijas o padres) van a ser manejadas, en general, como objetos no considerar un destructor virtual. Observe el siguiente trozo de código que aparece en el main del ejemplo (fichero main.cpp).

```
...
Ventana *ventana1 = new MiVentana1 (...);
Ventana *ventana2 = new MiVentana2 (...);
...
delete ventana1;
delete ventana2;
...
```

Si el destructor definido en la clase Ventana no fuera virtual, los dos *delete* del final siempre invocarían a aquél, en lugar del que figure definido dentro de las propias clases hijas (MiVentana1 y MiVentana2) con las posibles pérdidas de memoria que ello puede implicar.

CONFIGURACIÓN Y MACROS

Dentro del fichero "config.hh" se declaran, por un lado, un conjunto de macros que sirven de configuración del sistema y, por otro, la macro parametrizada "DENTRO_VENTANA" que advierte a la clase pantalla que el próximo dibujo se va a hacer dentro de una zona muy limitada de una ventana concreta pasada como parámetro.

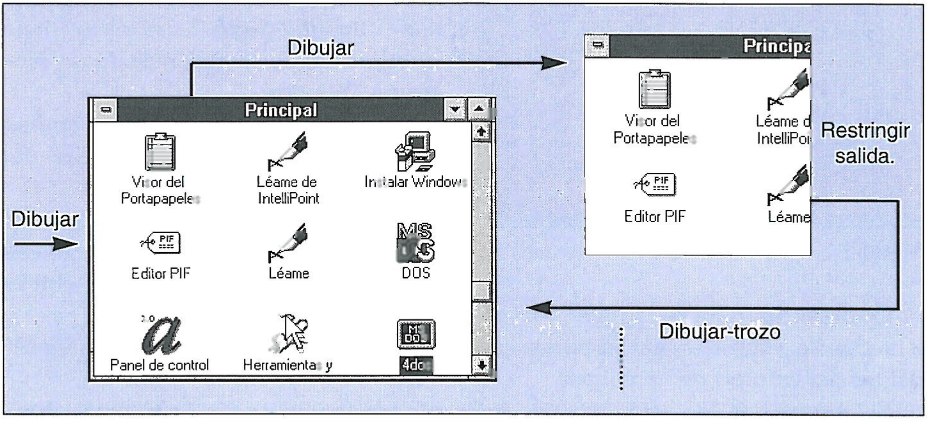


Figura 4: Conversación entre una ventana y sus zonas.

La clase NodoZona

```
// Clase NodoZona
// Realizado por Ignacio Cea Forniés
// Copyright de Sólo Programadores

        #if !defined (__ZONNODE_HH__)
        #define __ZONNODE_HH__

        extern "C" {
          #include <stdio.h>
        };

        class Ventana;
        class Objeto;

// Características de la clase
class NodoZona {
  protected :
    NodoZona *Anterior;
    NodoZona *Siguiente;
    int X1,Y1,X2,Y2;
    Ventana *Propietario;

    int numero_zonas (void);

  public :
   NodoZona (int x1,int y1,int x2,int y2,Ventana *pro-
pietario,
NodoZona *ant = NULL, NodoZona *sgt = NULL);
    ~NodoZona ();
    int x1 (void) const;
    int y1 (void) const;
    int x2 (void) const;
    int y2 (void) const;
    NodoZona *anterior (void) const;
    NodoZona *siguiente (void) const;
    Ventana *propietario (void) const;
    void borrar (void) const;
    void dibujar (void) const;
    void dibujar (const Objeto &) const;
};

// Métodos inline
inline int NodoZona::x1 (void) const { return (X1); }
inline int NodoZona::y1 (void) const { return (Y1); }
inline int NodoZona::x2 (void) const { return (X2); }
inline int NodoZona::y2 (void) const { return (Y2); }
inline NodoZona *NodoZona::anterior (void) const {
return (Anterior); }
inline NodoZona *NodoZona::siguiente (void) const
{ return (Siguiente); }
inline Ventana *NodoZona::propietario (void) const {
return (Propietario); }

        #endif

// Fin del fichero NodoZona.hh
// Versión 1.0
// 14 de Julio de 1995
```

Figura 5.

El destructor de ventanas

```
/////////
// Elimina la ventana. El sistema se reconstruye.
/////////
Ventana::~Ventana ()
{
// Recorre todas las zonas tapadas por esta venta-
na...
// ...dibujándolas
NodoZona *Zona = ZonasTapadas.primero ();
while (Zona != NULL) {
  Zona -> borrar ();
  Zona -> dibujar ();   // Implica limpiar fondo y pin-
tar...
  Zona = Zona -> siguiente (); // ...el interior de la
misma
}

// Busco el nodo de ventana que tiene ésta...
// ..y lo destruyo
NodoVentana *Nodo = Inferior;
while (Nodo != NULL && Nodo -> propietario () !=
this)
  Nodo = Nodo -> siguiente ();
if (Nodo == Inferior) Inferior = Inferior -> siguiente ();
if (Nodo == Superior) Superior = Superior -> anterior
();
delete Nodo;            // Se elimina así de la lista

// Regenera el sistema
zonas_vistas ();
zonas_ocultas ();
}
```

Figura 6.

El fichero Config hh

```
// Declaración de macros de configuración del siste-
ma
// Realizado por Ignacio Cea Forniés
// Copyright de Sólo Programadores

        #if !defined (__CONFIG_HH__)
        #define __CONFIG_HH__

        // Macros de configuración de pantalla
(VGA 16 colores)
        #define MAXX                    639
        #define MAXY                    479

        // Color de fondo del sistema
        #define COLOR_FONDOSYS          3
        #define COLOR_TEXTOSYS          11

        // Distintos tipos de eventos
        #define NADA                    0
        #define PULSARTECLA             1
        #define USUARIO                 2

        // Resultados de la gestión de un evento
        #define EVENTO_TRATADO          0
        #define EVENTO_DESCONOCIDO 1
        #define EVENTO_PADRE            2
        #define EVENTO_SALIR            -1000

        // Sitúa el punto de referencia dentro de
una ventana
        #define DENTRO_VENTANA(Vnt)
                                \
                Pantalla::mover_origen_a
(Vnt -> x1 (),Vnt -> y1 ())
        #endif
// Fin del fichero Config.hh
// Versión 1.0
// 14 de Julio de 1995
```

Figura 7.

Observe la figura 7 para ver el aspecto de esta, de vital importancia dentro del entorno de ventanas.

El lugar donde esta macro es empleada es, como ya habrá adivinado, dentro de los métodos dibujar situados en la clase NodoZona.

INICIALIZACIÓN AUTOMÁTICA

Antes de poder crear cualquier ventana, el entorno necesita dos cosas: Por un lado, inicializar el sistema gráfico y, por otro, crear una ventana inicial que haga de fondo del mismo. Ambas cosas han de ser logradas, lógicamente, sin que el programador tenga porque escribirlas. ¡Imagine lo feo que quedaría que la primera línea del main fuera un *initgraph* !. ¿Acaso tiene usted que hacer eso dentro de un programa "Windows"?.

La inicialización del entorno gráfico parece lógico incluirla dentro del constructor de la clase Pantalla. Sin embargo, para ejecutarlo hay, al menos, que crear un objeto de ese tipo lo cual puede conseguirse con la línea:

static Pantalla PANTALLA_SISTEMA;

situada en la parte final del fichero "Pantalla.cpp".

Para crear un fondo, es necesario en primer lugar crear una clase hija de Ventana (a la que se llamará "Fondo") y crear un objeto de ese tipo de forma similar a la vista anteriormente con la inicialización de la pantalla (fichero fondo.cpp).

De esta forma, las clases que forman el sistema (todas menos los hijos de Ventana) pueden meterse en una librería, que al ser enlazadas con cualquier otro programa inicializará de forma automática un entorno de ventanas.

COMPILAR EL EJEMPLO

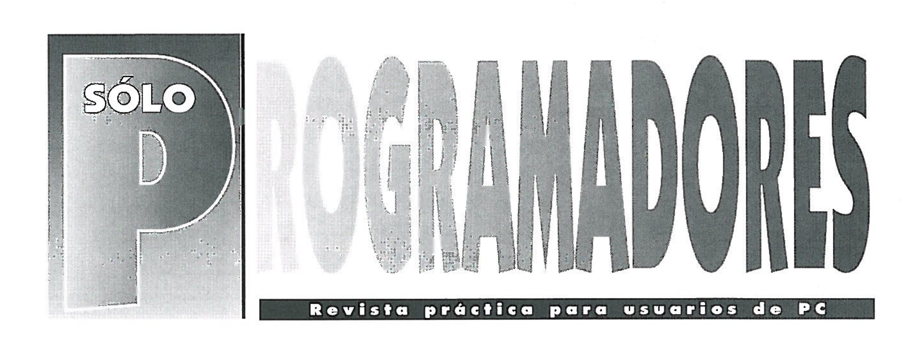
El ejemplo ha sido desarrollado íntegramente empleando el compilador "BorlandC++ 3.1". Emplear otro compilador no debería variar substancialmente ninguna de las clases empleadas salvo, si cabe, los métodos de dibujo empleados dentro de la clase pantalla que, evidentemente, no son estándar ANSI sino propios de la Borland. Un listado de los ficheros que integran el proyecto podrá hallarlo en la figura 8.

CORREO LECTORES

Para cualquier duda referente a los temas/artículos tratatos en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027

COMO SUSCRIBIRSE A



uscríbase enviando este cupón por correo o fax (91) 320.60.72, o llamando al teléfono (91) 741.26.62 Horario 9 a 14 y 15:30 a 18:30 h.

)eseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.

ıombre y apellidos...Domicilio...

'oblación..C.P ....Provincia.........................Telf...............................Profesión....................

'ORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Fecha de caducidad de la tarjeta.................Nombre del titular, si es distinto...

☐ Domiciliación bancaria.

Señor Director del banco ....

Población ....

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta

ͅ ahorro número...

recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L.

mo pago de mi suscripción a la revista SÓLO PROGRAMADORES.

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Marques de Portugalete 10, Bajo
28027 Madrid.

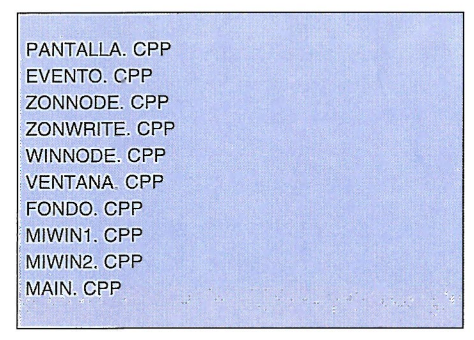


Figura 8: Los ficheros del proyecto ejemplo.

La ejecución del ejemplo requiere, además, que dentro del *path* actual figuren los ficheros "EGAVGA.BGI" y "SANS.CHR".

Seguramente, tendrá que cambiar el *path* para acceder a los ficheros *include* del sistema (..\INCLUDE) ya que el proyecto contempla los que existían en la máquina del autor.

El proyecto debe ser compilado empleando el modelo *large* y con las librerías gráficas enlazadas por defecto.

EL EJEMPLO

El ejemplo muestra dos ventanas dentro del sistema. La primera de ellas, de color verde y con una recta blanca de izquierda a derecha, puede moverse a través de la pantalla mediante las teclas "q", "a","s" y "z", además de emitir un pequeño zumbido si se pulsa "b". La otra, de color azul y con una raya blanca de derecha a izquierda", puede reescalarse mediante las teclas "o","k","l", y "," además de emitir un pequeño zumbido al pulsar "n". Para abandonar el sistema basta con presionar <ENTER>. Un aspecto de la pantalla se muestra en la figura 9.

Las clases que forman el sistema pueden meterse en una librería que se enlazarán con otro programa

UN CONSEJO

Es importante que compile el ejemplo con información de *debug* y que emplee la ejecución paso a paso para ver la forma de evolucionar del entorno y con ello adquirir un conocimiento más profundo del mismo.

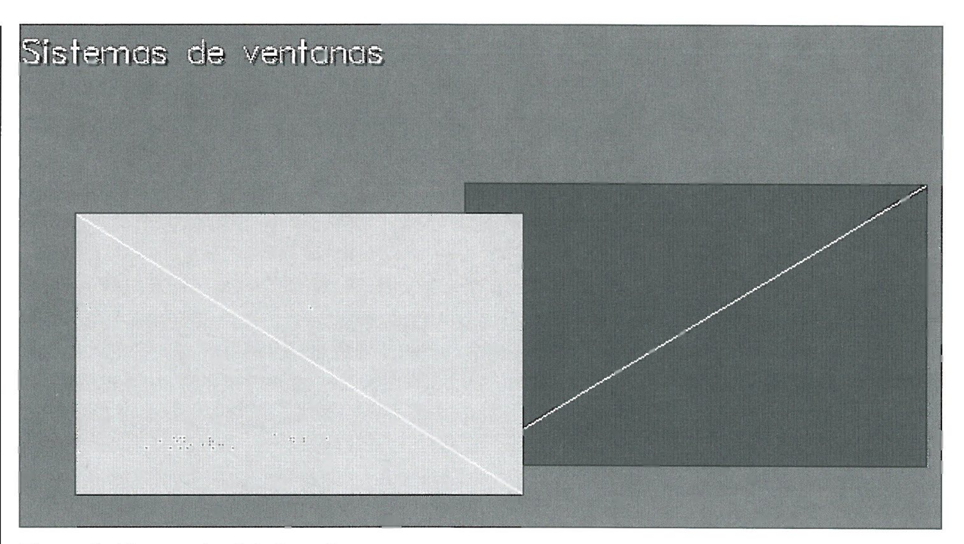


Figura 9: El aspecto del ejemplo.

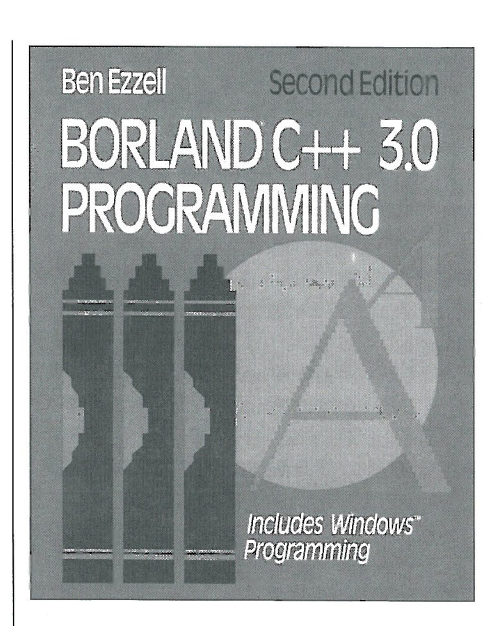
PROPUESTAS

Durante las últimas semanas se nos han quedado como es lógico algunos detalles en el tintero. Entre ellos cabe destacar el hecho de que las ventanas no pueden intercambiar su orden de aparición o el que los eventos contemplados sean simplemente el "ninguno" o "la pulsación de teclas". Le proponemos que modifique el código para lograr que las ventanas puedan ascender dentro del sistema o que se puedan aceptar eventos, como por ejemplo el movimiento del ratón o la llegada de un dato por el puerto serie. Será, sin duda, un interesante problema a resolver. Suerte.

LA PRÓXIMA ENTREGA

A partir del próximo artículo se comenzará una serie en la que se desarrollará un conjunto de clases genéricas que permitirá el desarrollo de juegos conversacionales, además de acompañarlo, claro está, con un completo ejemplo en el que se aúnen todas las técnicas vistas hasta el momento.

Esperamos que este apasionante tema sea, también, de su interés. Hasta entonces se recomienda la lectura de varios libros muy interesantes para completar el presente tema:



Bibliografía: Borland C++ 3.0 programming, Autor: Ben Ezzell.

"The Design and Evolution of C++"
autor: Bjarne Stroustrup
idioma: inglés
editorial: Addison-Wesley

"The Art and Science of C"
autor: Eric S. Roberts
idioma: inglés
editorial: Addison-Wesley

"Borland C++ 3.0 programming"
autor: Ben Ezzell
idioma: Inglés
editorial:Addison-Wesley

RAZONAR DE FORMA APROXIMADA

Juan José Samper

El saber de un especialista humano se almacena en la base de conocimientos de un sistema experto. La mayoría de los conocimientos pueden ser imprecisos. Esa incertidumbre provoca a su vez incertidumbre acerca de las conclusiones que obtiene el sistema. Por ello, es necesario controlar su propagación a lo largo del proceso de razonamiento. Existen varios modelos de incertidumbre. Puede ser probabilística, cuando el problema tiene un componente aleatorio, o de tipo posibilístico, cuando se toman decisiones a partir de un conjunto de creencias. La probabilidad es un pronóstico "a priori" sobre lo que se espera que salga (como el lanzamiento de un dado), y la posibilidad es una incertidumbre "a posteriori" (ejemplo: ¿cuántos huevos se puede comer?). No hay que confundir posibilidad con probabilidad: es posible que Miguel desayune 20 huevos, pero es poco probable. Dentro de los modelos, en cuanto a medida, se tienen aquellos que evalúan la incertidumbre con un valor numérico (se comerá 10 huevos con grado de creencia de 0.3) y los que la evalúan con un valor lingüístico (regular, difícil, imposible,...). El tratamiento de la incertidumbre es complejo. Es lo que hace que se pueda hablar de sistema con inteligencia: el razonamiento del ser humano se considera inteligente por utilizar conocimiento del que no dispone "a priori".

En general, se necesitará una herramienta de combinación de la incertidumbre de varios sucesos simples, y un método de propagación de la incertidumbre en una cadena de decisiones y soluciones.

FUENTES DE INCERTIDUMBRE

Normalmente, el conocimiento es incierto o inexacto debido a que:

- Existen datos aleatorios, como, por ejemplo, el peso de la gente.
- Puede haber datos insuficientes. Ejemplo, no se puede determinar el grado de apendicitis de un enfermo sin operarle (o sin una muestra).
- El conocimiento disponible se expresa de forma "vaga". Ejemplo, la palabra grande no tiene significado por sí sola sin incluirla en un contexto, ya que no es lo mismo hablar de un pájaro grande que de un hipopótamo grande.
- Hay situaciones en las que el conocimiento está basado sólo en presentimientos.
- La fuente de conocimiento puede no ser totalmente fiable. En estos casos, es necesario incorporar a la representación del conocimiento una medida de incertidumbre.

INFERENCIA E INCERTIDUMBRE EN REGLAS DE PRODUCCIÓN

Los sistemas expertos basados en reglas modelan el conocimiento mediante reglas de producción con la forma:

SI e ENTONCES h

donde e es la evidencia (combinación de condiciones simples relacionadas por operadores Y y O) y h es la hipótesis (conjunción de conclusiones).

La regla tendrá el siguiente significado: "Si la evidencia e ha sido observada, entonces la hipótesis h se confirma como cierta". En la práctica, sin embargo, una hipótesis rara vez es afirmada con absoluta certeza por la observación de una evidencia cierta. Por ello, se asocia una medida de incertidumbre x con la hipótesis de la regla de producción, que indica el grado en el que h es confirmada por la observación de la evidencia. Ejemplo:

SI e_1 Y e_2 ENTONCES h_x

La mayoría de los conocimientos humanos son imprecisos. Un aspecto básico en un sistema es cómo manejar adecuadamente la incertidumbre contenida en su base de conocimientos.

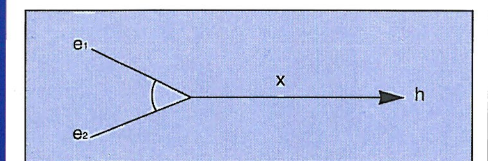


Figura 1. Red de inferencia con incertidumbre.

Esta regla puede representarse de forma gráfica mediante una red de inferencia, tal y como se muestra en la figura 1.

Para razonar con la incertidumbre existirá un mecanismo de inferencia, compuesto por:

- Una función de propagación de evidencias inciertas, que calcula la medida actual de incertidumbre asociada a la hipótesis h, de acuerdo con todas las evidencias anteriores. Ejemplo: la red de inferencia de la figura 2 representa la situación en la cual la hipótesis e ha sido confirmada en grado x por una evidencia anterior e'.

La medida de incertidumbre que se asociará con h dependerá tanto de y como de x.

- Unas funciones de combinación que calculen la medida de incertidumbre para una evidencia compuesta (condiciones unidas por operadores Y/O). Como ejemplo, dada la regla:

SI e_1 Y e_2 ENTONCES h_z

donde e_1 y e_2 son confirmadas en grado x e y de acuerdo con una evidencia anterior e'. Se tendrá que calcular una medida de incertidumbre para la evidencia compuesta. En la figura 3 puede verse la red de inferencia de la regla.

- Una función de combinación que calcule la medida de incertidumbre para una hipótesis h mediante las medidas parciales de incertidumbre de diferentes líneas de razonamiento. Es decir, cuando se tienen dos reglas que concluyen en la misma hipótesis. Un ejemplo de este caso puede verse en la figura 4.

TEORÍA DE LA PROBABILIDAD

La probabilidad se define como el grado de creencia "a priori" acerca de la realización o no de un fenómeno aleatorio.

Así pues, la probabilidad no mide la vaguedad de un suceso sino la creencia o no del suceso.

La visión de la probabilidad puede ser:

- Objetivista, en donde se considera que la probabilidad de un suceso es una cualidad de dicho suceso.

- Subjetivista, que considera que la probabilidad de un suceso no es inherente a él, sino a la persona que lo evalúa.

Si todos los acontecimientos tienen la misma probabilidad de salir, entonces la probabilidad de un suceso específico X será:

$$P(x)=\frac{I}{N}$$

en donde N es el número de sucesos posibles. Ejemplo: la probabilidad de que salga 4 al tirar un dado es de 1/6 porque hay 6 resultados posibles y todos son igual de probables.

Toda probabilidad tiene un valor comprendido entre 0 y 1 ambos inclusive. Un suceso con probabilidad 1 es una certeza. Un suceso con probabilidad 0 es considerado como imposible. Algunas de las herramientas de medición que proporciona la teoría de la probabilidad son:

1. Si se tiene *P(A) y P(B)* y A y B son sucesos independientes, entonces $P(A \cap B)=P(A)*P(B)$. (*P(A)* denota la probabilidad del suceso A, y es un número entre cero y uno).

2. La probabilidad del suceso complementario es igual $P(\neg A)$ *a* $1-P(A)$.

3. $P(A \cup B)=P(A)+P(B)-P(A \cap B)$

La regla de Bayes permite calcular la probabilidad de que se cumpla una hipótesis H habiéndose observado una evidencia E. Se denota P(H/E), y su valor se obtiene según la fórmula:

$$\frac{P(E/H) * P(H)}{P(E)}$$

Ejemplo: Se tiene la evidencia E= fiebre y la hipótesis H= bronquitis, y se sabe que P(E/H)= 0.95 (es decir, que la bronquitis produce fiebre en el 95% de los casos). Entonces calculando P(H/E) se podrá saber la probabilidad de que se tenga bronquitis dado el hecho "tener fiebre".

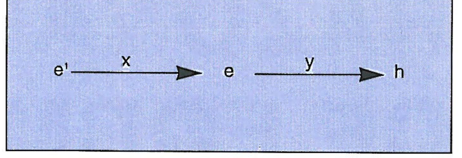


Figura 2.

Durante los años 60-70 se usó la teoría de la probabilidad para abordar el problema de la incertidumbre, pero daba problemas debido a que:

- Es una medida aleatoria, un pronóstico "a priori". Esto hace difícil el obtener valores exactos para las probabilidades, y en muchos casos sólo se cuentan con estimaciones subjetivas.

- Requiere mucho conocimiento para dar una probabilidad, lo cual implica un mal tratamiento de la incertidumbre. Ejemplo: Dada una urna que contiene bolas de distintos colores, es necesario saber el número de bolas de cada clase para definir la probabilidad de que salga una determinada. Si no se conoce cuántas hay de cada, se utiliza el Principio de la Razón Insuficiente: todas tienen la misma probabilidad de salir.

- Hay casos que no resuelve.

- La aplicación de la fórmula de Bayes conlleva un costo computacional elevado (hay que realizar muchas operaciones de multiplicación y de división).

Para intentar paliar esto último, Turing expresó la fórmula de Bayes en función de "soportes":

$$SOP(A)=\frac{P(A)}{P(\neg A)} \Rightarrow P(A)=\frac{SOP(A)}{1+SOP(A)}$$

La fórmula queda entonces como: SOP(H/E)=SOP(H)*LS(E), en donde LS es la suficiencia lógica definida por:

$$LS\ (E)=\frac{P(E/H)}{P(E/\neg H)}$$

Ejemplo: Calcular la probabilidad de la hipótesis x habiéndose observado las evidencias A y B. La probabilidad a priori de x es 0.3. LS para A es 20 y LS para B es 100.

$$Sop(x)=\frac{0.3}{(1-0.3)}=0.43$$

$$LS\ de\ A=20 \Rightarrow$$

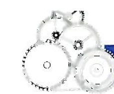
$$Sop(x/A)=Sop(x)*LS(A)=0.43*20=8.6$$

$$P(x/A)=\frac{Sop(x/A)}{1+Sop(x/A)}=\frac{8.6}{(1+8.6)}=0.9$$

$$LS\ de\ B=100 \Rightarrow$$

$$Sop(x/A\ Y\ B)=Sop(x/A)*LS(B)=8.6*100=860$$

$$P(x/A\ Y\ B)=\frac{Sop(x/A\ Y\ B)}{1+Sop(x/A\ Y\ B)}=\frac{860}{1+860}=0.99$$



TEORÍA DE LOS FACTORES DE CERTEZA

También llamada Teoría de la Certidumbre, fue ideada por Shortliffe y Buchanan (1975) para tratar la incertidumbre en MYCIN (sistema experto en enfermedades de origen infeccioso), intentando subsanar los fallos asociados a la teoría de la probabilidad. Trata de medir la confianza que se le otorga a una conclusión dada una evidencia.

A cada declaración o hecho S se le asocia una medida de certeza C(S). Esta medida pertenece al intervalo [-1, 1] y se define:

C(S)= 1 si se sabe que S es cierto.

C(S)= -1 si se sabe que S es falso.

C(S)= 0 si no se sabe nada acerca de S.

Los valores intermedios indicarán una medida de la certidumbre o incertidumbre de S.

A cada regla se le asociará un factor de certeza CF(H/E) que es una medida de su fiabilidad, y representa la proporción de aumento o disminución de certeza sobre la hipótesis H al observar la evidencia E. Estos valores también pertenecen al intervalo [-1, 1], y significarán:

Si CF(H/E)= 1, se cree sin restricciones que se cumple H.

Si CF(H/E)= -1, se descarta sin restricciones que se pueda cumplir H.

Si CF(H/E)= 0, se tiene que la observación de E no condiciona para nada lo que pueda pasar con H.

Si 0 < CF(H/E) < 1, entonces hay evidencias que sostienen a H, es decir, aumentará la creencia de H (C(H)).

Si -1 < CF(H/E) < 0, entonces hay evidencias que rechazan a H, disminuirá la creencia de H.

En general, las reglas tendrán el siguiente formato:

SI A ENTONCES B con CF=x

Ejemplo:

SI la moto anda
Y la batería está bien
ENTONCES la moto arrancará de nuevo
con CF= 0.99

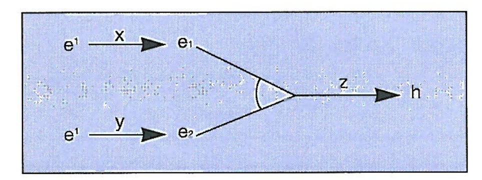


Figura 3. Regla con condición compuesta.

Inicialmente el valor de certeza para cada proposición o hecho es 0. Dada una regla A→B con CF=x y con A cierta, el nuevo valor de certeza de B es igual a x.

Para propagar la incertidumbre, se tienen en cuenta dos casos:

1º Que la condición A de una regla sea verdadera (C(A)= 1). Entonces la regla se puede utilizar para calcular un nuevo valor de certeza para su conclusión (C(B/A)) de la siguiente forma:

$$C(B/A) = C(B) + (1 - C(B)) * CF \quad \text{si } C(B) \text{ y } CF \text{ son } \geq 0$$
$$C(B/A) = C(B) + (1 + C(B)) * CF \quad \text{si } C(B) \text{ y } CF \text{ son } \leq 0$$

En cualquier otro caso se utilizará la ecuación:

$$C(B/A) = \frac{C(B) + CF}{1 - \text{mínimo}\{|C(B)|, |CF|\}}$$

(las barras verticales | | significan valor absoluto)

2º Que la condición de la regla sea incierta (C(A)< 1). Esto significa que A es un factor de incertidumbre inicialmente en la base de datos o que A es conclusión de otra regla. En ambos casos, la certidumbre de la conclusión de la regla debe reducirse. Un enfoque es multiplicar el factor de certidumbre CF de la conclusión de la regla por la certidumbre de la condición C(A), si ésta es positiva, e ignorar la regla si C(A) es negativa o no supera cierto umbral.

Si una regla tiene la condición compuesta (varias condiciones unidas por operadores Y u O), se necesita calcular su valor de certeza global. Para ello se utilizan las fórmulas:

C(A Y B)= mínimo{C(A), C(B)}

C(A O B)= máximo{C(A), C(B)}

Ejemplo, suponiendo las reglas R1, R2, R3 y los valores iniciales de certeza dados para las declaraciones, se quiere saber el valor final de certeza para F.

R1: SI A O B ENTONCES C con CF_1= 0.8

R2: SI D ENTONCES E con CF_2= 0.7

R3: SI C Y E ENTONCES F con CF_3= 0.9

C(A)=1, C(B)=0.6, C(D)=0.9, C(F)=0.2, C(C)=0, C(E)=0

Se calcula el valor de certeza para la condición compuesta de R1:

C(A O B)= máximo{C(A), C(B)}= máximo{1, 0.6}= 1

Se calcula el nuevo valor de certeza de C:

C(C/(A O B))= C(C)+(1-C(C))*CF_1= 0+(1-0)*0.8= 0.8

Se realiza de esta manera debido a que la certeza de la condición es 1, es decir, se cree totalmente cierta.

Se calcula el nuevo valor de certeza para E:

C(E/D)= CF_2*C(D)= 0.7*0.9= 0.63

Se realiza así debido a que la certidumbre de la condición D es menor que 1.

Se calcula el valor de certeza para la condición compuesta de R3:

C(C Y E)= mínimo{C(C), C(E)}= mínimo{0.8, 0.63}= 0.63

Finalmente, el valor de certeza para F será:

C(F/(C Y E))= CF_3*C(C Y E)= 0.9*0.63= 0.567

Algunas de las ventajas de esta teoría son:

- Tiene una extensa aplicación en los sistemas expertos.
- El resultado de los valores de certeza siempre oscila entre 1 y -1 y está bien definido.
- Proporciona una forma de realizar estimaciones subjetivas en los casos en que los valores exactos de las probabilidades no puedan obtenerse.
- Si se aplican dos reglas contradictorias, siendo sus certezas iguales, los efectos se cancelan.

PROBABILIDAD SUPERIOR E INFERIOR

La teoría de la probabilidad puede evaluar un suceso cuando éste se conoce

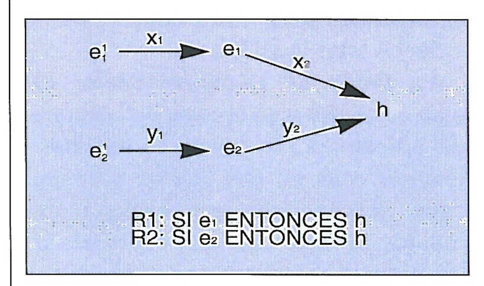


Figura 4. Red de inferencia para 2 reglas que concluyen en una misma hipótesis.

completamente. Si no, la probabilidad obliga a emplear la teoría de la equiprobabilidad de Laplace. Esto puede llegar a deformar el problema.

Para resolver el inconveniente, se utiliza la teoría de la probabilidad superior e inferior: para cada suceso existe una probabilidad superior y otra inferior a la verdadera probabilidad del suceso.

Lo que se hace es evaluar la incertidumbre de un suceso A mediante un intervalo que indica lo menos y lo más que se apuesta por la ocurrencia de A con la información de que se dispone. Se representa:

$$A:[P_*(A), P^*(A)]$$

en donde $P_*(A)$ representa la probabilidad del peor caso y $P^*(A)$ representa la probabilidad del caso más favorable.

TEORÍA DE LA EVIDENCIA DE DEMPSTER-SHAFER

En 1967, Dempster introduce un modelo de representación de incertidumbre inspirado en los conceptos de probabilidad inferior y superior.

Posteriormente, Shafer trabaja sobre el modelo y lo modifica, dando lugar a lo que hoy se conoce como teoría de la evidencia, siendo ésta una de las herramientas más efectivas para trabajar con la incertidumbre.

Esta teoría está basada en la idea de establecer un número entre cero y uno para indicar el grado de creencia de una proposición o hecho. En ella se tiene en cuenta la ignorancia de un suceso, en contraste con la teoría de la probabilidad, donde $P(A) + P(\neg A)=1$ para cualquier proposición A. Eso no corresponde necesariamente a una descripción del mundo real porque no se ha tenido en cuenta la ignorancia. Si no se dispone de evidencias a favor o en contra de A, entonces es apropiado asumir que ambos grados de creencia, para una evidencia de A y de $\neg A$, son igual a cero.

La teoría de Dempster-Shafer es algo compleja y se escapa del alcance del artículo, por lo que no se va a profundizar más en ella. Como ventaja, cabe señalar que proporciona un marco común donde representar informaciones de diversos tipos, especialmente probabilidades y posibilidades.

TEORÍA DE LOS CONJUNTOS DIFUSOS

La introdujo L.A. Zadeh en 1965. A partir de ella, se construye la lógica difusa, que está diseñada para tratar conceptos vagos.

Con esta teoría se pueden tratar universos finitos e infinitos.

Se habla del grado de vaguedad que tienen las cosas, de incertidumbre "a posteriori". En todos los modelos vistos hasta ahora, una proposición puede ser verdadera o falsa. La vaguedad es la propiedad de no ser estrictamente verdadera o falsa.

Sea X un conjunto llamado universo o referencial. Sus elementos se denotan por x. Los miembros de un subconjunto clásico $A \subseteq X$ pueden verse como una función característica $\mu_A(x)$ y su resultado estará incluido en el conjunto {0, 1}, de forma que:

$$\mu_A(x) = 1 \text{ si } x \in A$$
$$\mu_A(x) = 0 \text{ si } x \notin A$$

A {0, 1} se le llama conjunto de evaluación.

Si el conjunto de evaluación pasa a estar formado por los infinitos elementos comprendidos en el intervalo cerrado [0, 1], a A se le llama subconjunto difuso y $\mu_A(x)$ será el grado de pertenencia de x al subconjunto A. Con esto, se observa que:

- Cuanto más se acerca $\mu_A(x)$ a 1, más evidente es la pertenencia del elemento x al conjunto A.
- Las fronteras del conjunto A no están bien definidas.

Con esta teoría se puede realizar un razonamiento aproximado, es decir, obtener consecuencias vagas a partir de conocimientos vagos.

TEORÍA DE LA INCIDENCIA DE BUNDY

Es un método que facilita el cálculo de la probabilidad de una proposición de forma sistemática. Esta probabilidad estará basada en un conjunto de situaciones denominadas incidentes.

Si se nota por ϕ al conjunto de todos los incidentes, la incidencia de la proposición A con respecto a ϕ será el subconjunto de ϕ que contiene a todos los incidentes para los cuales A es cierta.

De manera más formal:

Sea $\phi = \{i_1, ..., i_n\}$ un conjunto de incidentes.

Sea A un suceso $A=\{x / x \in \phi\}$.

La incidencia de una proposición A se define:

$$i(A)= \{i_j / i(A)_j = \text{verdadera}\}$$

Esta incidencia se puede expresar como un vector:

$$i(A) \equiv (e_1 \dots e_n)= e(A)$$

cumpliendo que $e_j=1$ si $i_j \in i(A)$

$e_j=0$ si $i_j \notin i(A)$

De esta forma, los incidentes pueden representarse mediante cadenas de bits.

Propiedades:

$$- \ i(A \cup B) = i(A) \cup i(B)$$

. Si se trabaja con vectores, será la suma booleana de e(A) y de e(B).

$$- \ i(A \cap B) = i(A) \cap i(B)$$

. Si se trabaja con vectores se tendrá que hacer el producto término a término de e(A) y de e(B).

$$- \ i(\neg A) = \phi - i(A).$$

Ejemplo: Se lanza un dado y una moneda.

Se tendrá ϕ = {(dado, moneda)}

ϕ tendrá, por tanto, 12 incidentes:

(1, cara) (1, cruz)
(2, cara) (2, cruz)
(3, cara) (3, cruz)
(4, cara) (4, cruz)
(5, cara) (5, cruz)
(6, cara) (6, cruz)

El incidente A: "Salir 2 en el dado" será:

i(A)= 00 11 00 00 00 00

Los unos en esa posición representan los pares (2, cara) y (2, cruz).

De igual manera, el incidente "Salir cara en la moneda" sería:

i(A)= 10 10 10 10 10 10

De esta forma, se llama $i(A)_j$ al valor del bit que ocupa la posición j, y $P(\phi_j)$ a la probabilidad del correspondiente incidente. Se tendrá que:

$$P(A) = P(i(A)) = \sum_j i(A)_j * P(\phi_j)$$

Esto hace que se pueda tener un método sistemático para calcular las

probabilidades. Además cuando los sistemas son equiprobables, se reduce a contar bits y multiplicarlos.

POR FIN, EL SOFTWARE

Un sistema experto puede dar un buen consejo acerca de un problema. Para que sea verdaderamente útil, el sistema debería informar también acerca de la probabilidad de que dicho consejo sea correcto. Esta capacidad es la que se le ha añadido al programa que acompaña al artículo.

El código fuente se ha elaborado en Turbo C y se encuentra en el fichero SISTEMA.C. El programa ejecutable es el fichero SISTEMA.EXE. Como anteriormente, se adjunta una pequeña base de conocimiento FRUTAS para que el lector pueda experimentar.

El sistema experto de este artículo tendrá asociada una probabilidad con cada información que conozca. La base de conocimientos se ha alterado añadiendo otro campo llamado proba a la estructura condicion. Ese campo contendrá la probabilidad asociada a cada condición o hecho.

Al introducir la condición de una regla, el programa pedirá también la probabilidad de que esa condición esté asociada con la acción o hecho que desencadena. Por ejemplo, se puede pensar que el síntoma tos tiene un 45% de probabilidad de aparecer en un resfriado. La función basedereglas() incluirá tanto a las condiciones como a las probabilidades asociadas según la acción.

Las funciones comprueba() y no_es_condicion() se han modificado para que puedan manejar respuestas negativas que rechazan la hipótesis. En el sistema probabilístico actual, el rechazo sucederá sólo si la probabilidad de la condición supera un umbral determinado: 50%. Este porcentaje puede cambiarse, ya que su elección ha sido arbitraria. Cuando la probabilidad de encontrar cierta condición es suficientemente baja, el no tenerla no es motivo suficiente como para rechazar la conclusión que de ella deriva.

Existe una función nueva CalcuPro() que calcula la probabilidad de que la conclusión final del sistema sea correcta. Esto puede hacerse siguiendo el modelo de probabilidad clásica:

```
CalcuPro (float PCA)
{
PRO= PRO * PCA
}
```

(PCA es la probabilidad de la condición actual y PRO es una variable global que contiene la probabilidad de que la solución sea correcta).

Este modelo es inusual, ya que no funciona correctamente manejando información con incertidumbre.

En la teoría de la evidencia se tiene en cuenta la ignorancia de un suceso

También puede usarse un método que considere que la probabilidad de la solución sea la de la probabilidad más baja asociada a una de las condiciones:

```
if (PRO > PCA) PRO= PCA
```

Este método se utiliza cuando se quiere considerar la combinación de todas las condiciones.

O considerando la probabilidad más alta:

```
if (PRO < PCA) PRO= PCA
```

Este último método se empleará cuando se quiera usar cualquier condición para desencadenar la acción o conclusión. La solución reflejará la probabilidad de la evidencia más convincente. El programa SISTEMA utiliza esta filosofía.

EL SISTEMA DE AYUDA DE WINDOWS 3.1

Enrique Castañón

Queramos o no, más tarde o más temprano, tendremos que pasar por el aro de Microsoft Windows. La ayuda de Windows está basada en el hipertexto. El conocimiento de éste se hace imprescindible para todo programador que se precie.

Los soportes convencionales para transmitir la información como el papel, los audiovisuales, televisión, radio, etc. no permiten la interactividad por parte del receptor. En éstos, la información está previamente estructurada de forma secuencial o lineal. Como mucho podemos acceder a una parte de la información mediante un índice. La aparición del ordenador abrió nuevas posibilidades. En 1965 Ted Nelson definió lo que podía ser una nueva forma de escribir y presentar la información, como alternativa al método convencional lineal, a lo que llamó hipertexto.

Podemos definir el hipertexto como una forma no lineal o tridimensional de presentar la información textual. Es el propio texto mediante hiperenlaces el que nos permite navegar a través de la información. Estos hiperenlaces representan asociaciones conceptuales entre palabras, y otras partes de la información. De esta manera el receptor de la información deja de ser un sujeto pasivo, pudiendo interactuar con la información, siguiendo una secuencia no predeterminada.

Cuando además de texto, la información está compuesta por imágenes, animaciones, sonido, etc. estamos hablando de hipermedios. Tal vez este término a sido devorado por el popular "multimedia". Muchas de las enciclopedias multimedia que se comercializan actualmente, son ejemplos claros de hipermedios, aunque comercialmente hablando sea mejor utilizar "Multimedia Interactivo". No debemos confundir que; una aplicación multimedia no siempre es hipermedio, sin embargo, un hipermedio siempre es multimedia e interactivo. La diferencia viene dada por la forma de navegar a través de la información. Aquella que implemente métodos de acceso tridimensional, podrá considerarse hipermedio.

El hipertexto y los hipermedios son útiles allí donde haya una gran cantidad de información: enciclopedias, enseñanza por ordenador, documentación on-line, etc. La documentación on-line será la que nos ocupe en esta serie de artículos que comienzan, aunque no será la única. El hipertexto puede ser útil para distribuir productos literarios sobre cualquier tema. Como ejemplo, en una próxima entrega, construiremos un hipertexto con esta serie de artículos.

EL SISTEMA DE AYUDA DE WINDOWS 3.1

En esta primera entrega daremos un repaso a los fundamentos de los archivos de ayuda para Windows (HLP). Más adelante trataremos el uso de macros, la inclusión de gráficos, y la creación de librerías de enlace dinámico (DLL), para suplir las carencias del sistema de ayuda de Windows.

El sistema de ayuda de Windows supone un avance importante en cuanto a la unificación de criterios para creación de ayudas. Uno de los costes ocultos más importantes del software es, sin lugar a dudas, el aprendizaje. Los usuarios deben pasar muchas horas aprendiendo el funcionamiento de una aplicación. Por tanto, documentar bien los programas, no sólo facilitará las cosas al usuario, sino

que también hace el software más rentable.

HERRAMIENTAS Y UTILIDADES PARA CREAR AYUDAS

Se empezará por saber lo que necesitamos para crear un archivo de ayuda:

- *WINHELP.EXE:* es la aplicación que ejecuta los ficheros de ayuda. Posee un archivo de ayuda propio, que no se debería dejar de leer. Esta aplicación viene en el paquete de Microsoft Windows, sin embargo algunos paquetes de programación la entregan también.
- Un procesador de texto capaz de guardar los archivos en formato RTF (*Rich Text Format*), como puedan ser: Microsoft Word o WordPerfect por nombrar algunos. Los ejemplos que incluimos están creados con Microsoft Word 6.

- *HC31.EXE:* compilador de Microsoft para los archivos de proyectos HPJ. El archivo de proyecto deberá estar en texto ASCII. Esta aplicación viene en cualquier paquete de programación para Windows.

Además de estas herramientas básicas, suplementariamente se debería disponer de las siguientes:

- *Microsoft Hotspot Editor (SHED.EXE):* esta utilidad crea archivos de hipergráficos (SHG). Viene con cualquier paquete de programación para Windows.

- *Microsoft Multiple Resolution Bitmap Compiler (MRBC.EXE):* combina varios archivos gráficos en un solo archivo (MRB) con diferentes resoluciones. El sistema de ayuda se encarga de mostrar el que se adapte a la resolución del monitor. Viene con cualquier paquete de programación para Windows.

- La ayuda de Windows da soporte a archivos gráficos BMP y WMF. No estaría de más disponer de una aplicación gráfica que soporte estos formatos.

LOS ARCHIVOS RTF

Un archivo de ayuda está compuesto por una serie de temas a los que se accede mediante hiperenlaces. Cada tema puede estar en un archivo RTF propio, o pueden estar, separados mediante saltos de página, en un único archivo RTF. Cada tema debería describir un aspecto determinado de la aplicación, sin mezclar dos aspectos distintos en un mismo tema.

Los temas no tienen requerimientos mínimos, podrían estar vacíos, y no visualizaríamos ningún mensaje de error durante la compilación. Dada la poca utilidad de un tema de estas características, se debería incluir en cada tema: un título, un identificador único del tema, y el texto o cuerpo del tema con los hiperenlaces o referencias a otros temas.

Cuando nos hemos referido a un título, no entendíamos por tal, el que aparece en la primera línea del texto, sino al que podemos definir mediante el símbolo de nota al pie ($). Que aunque no es obligatorio, suele ser igual al que aparece en la primera línea del texto. Este título es el que se muestra en la lista del botón Historial, y en la lista Ir a del cuadro Buscar de la Ayuda.

El identificador del tema se define mediante el símbolo de nota al pie (#). Este identificador es el que utilizaremos para crear hiperenlaces.

Dentro del texto podemos crear hiperenlaces con el subrayado doble de una o varias palabras, inmediatamente después, y con el formato de texto oculto, indicaremos el identificador del tema al que se quiere saltar.

EL ARCHIVO DE PROYECTO HPJ

Este archivo único, es el que utiliza el compilador HC31.EXE para generar un archivo (HLP). Debe contener como mínimo la sección [FILES] con la lista de archivos RTF. El nombre de este archivo será el que tenga el archivo final (HLP), esto es, un archivo AYUDA.HPJ dará como resultado un archivo AYUDA.HLP.

En la sección [MAP] se les puede dar un valor a los identificadores de cada tema. De esta manera la aplicación puede pasar este valor a la ayuda para presentar un tema determinado.

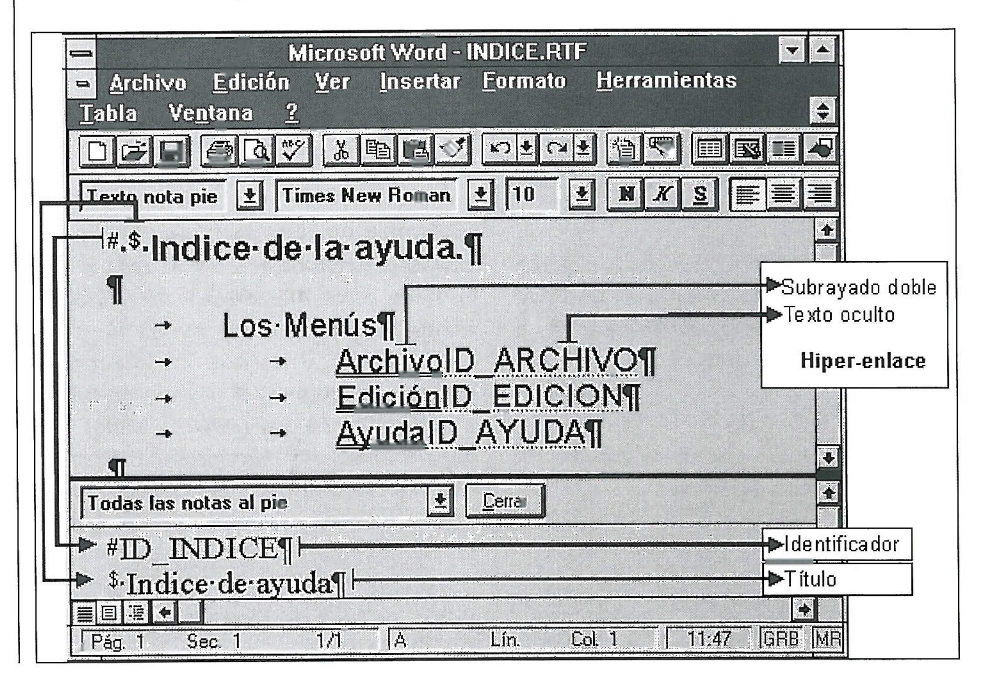
Lo mínimo para crear un archivo de ayuda es: un archivo RTF (aunque esté vacío), y un archivo de proyecto HPJ con la sección [FILES].

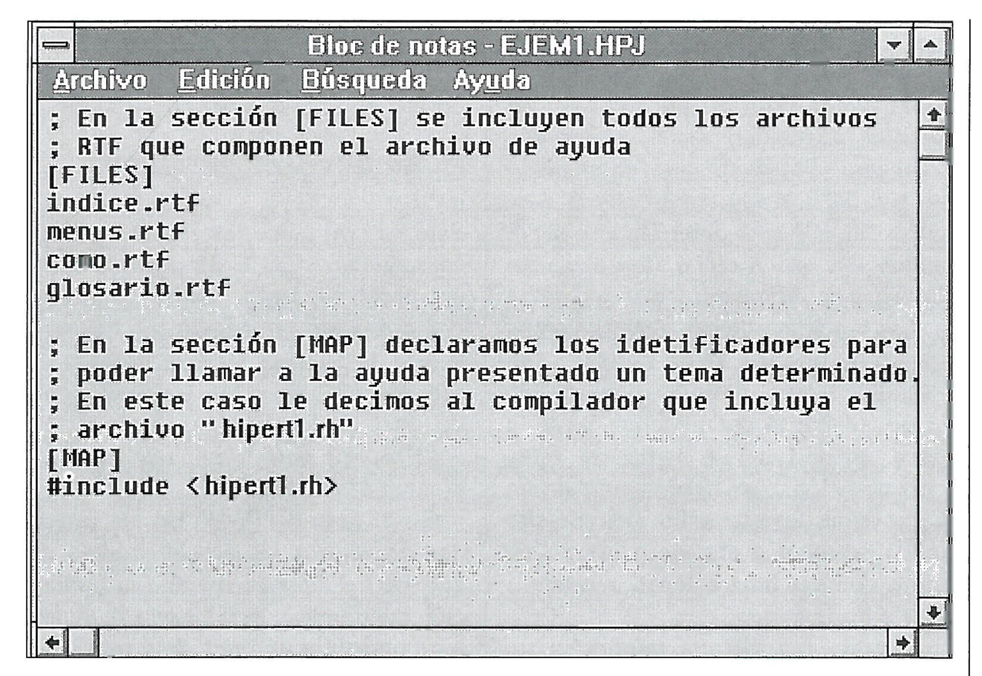
ENLACE CON LA APLICACIÓN

El API de Windows posee una función que permite enlazar la aplicación con el archivo de ayuda. Se trata de la función WinHelp. La sintaxis de esta función es la siguiente:

BOOL WinHelp(hwnd , lpszHelpFile , fuCommand , dwData)

El primer parámetro es el manejador (handle) de la ventana que llama a





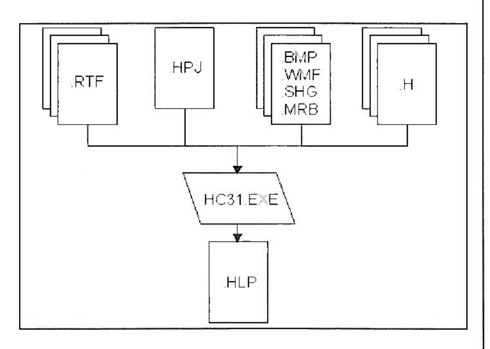
Bloc de notas - EJEM1.HPJ

Archivo Edición Búsqueda Ayuda

```
; En la sección [FILES] se incluyen todos los archivos
; RTF que componen el archivo de ayuda
[FILES]
indice.rtf
menus.rtf
como.rtf
glosario.rtf

; En la sección [MAP] declaramos los idetificadores para
; poder llamar a la ayuda presentado un tema determinado.
; En este caso le decimos al compilador que incluya el
; archivo "hipert1.rh"
[MAP]
#include <hipert1.rh>
```

la ayuda. El segundo parámetro es el nombre del archivo de ayuda. El tercer parámetro define de que manera se



abre el archivo de ayuda. Y el cuarto parámetro depende del valor que tenga fuCommand.

Algunos de los valores posibles para fuCommand son los siguientes:

- *HELP_INDEX* presenta el índice del archivo de ayuda, el valor de dwData debe ser 0.

- *HELP_CONTEXT* presenta el tema de la ayuda que se indique en dwData, este valor es el que se ha definido en la sección [MAP] del archivo de proyectos (HPJ).

- *HELP_QUIT* cierra la ayuda, el valor de dwData debe ser 0.

ORGANIZACIÓN DEL TRABAJO

Lo primero que se debería hacer antes de nada, es un esquema con el contenido de la ayuda. El esquema debería consistir en una jerarquía con el índice de contenidos en la punta:

Contenido
- *Los Menús*
  - *Archivo*
    - *Nuevo*
    - *Abrir*
    - *Guardar*
    - *Guardar Como*
    - *Salir*
  - *Edición*
    - *Copiar*
    - *Pegar*
  - *Ayuda*
    - *Indice*
    - *Uso de la Ayuda*
- *Como...*
    - *Abrir un archivo*
    - *Copiar texto*
    - *Pegar texto*
- *Glosario*

Se debe incluir en la ayuda todo lo que el usuario espera encontrar en ella. Si tiene poca experiencia en Windows échele un vistazo a la ayuda de otras aplicaciones, y busque los puntos en común. En general el usuario accederá a la ayuda por dos motivos, saber que hace un comando en particular o, saber como se realiza una determinada tarea.

El siguiente paso sería la creación de los texto (archivos RTF) y el archivo de proyectos (HPJ). Para finalizar se compila el archivo de proyecto con la utilidad HC31.EXE.

CONSEJOS PARA LA CREACIÓN DE AYUDAS

No desviarse de las convenciones para archivos de ayuda. Una ayuda, es eso precisamente, y no un sofisticado sistema de complicar la vida al usuario. Resultaría paradójico que el usuario tuviese que invertir más tiempo en aprender a utilizar la ayuda que en aprender el contenido de la misma.

La ventana que muestre la información debe ser clara y ordenada. No abuse de los colores ni de distintos tipos de fuentes.

Utilice las fuentes que se incluyen en el paquete original de Windows: Arial, Courier y Times New Roman. Si utiliza otras, el usuario final podría no tenerlas instaladas en su sistema.

Cuidado con el tamaño de las fuentes. Si está creando su archivo de ayuda con una configuración de pantalla de 1024x768, tal vez el tamaño de la letra le parezca muy pequeño. Tenga en cuenta que el usuario podría tener configurada su pantalla a 640x480, tal vez a éste último las letras le parezcan muy grandes. Escoja un tamaño de fuentes de compromiso, teniendo en cuenta que la mayoría de los usuarios tienen una configuración de 640x480, aunque cada vez está más extendida la de 800x600.

EL PROGRAMA DE EJEMPLO

Se incluye un ejemplo de ayuda sensible al contexto. En esta entrega se ha explicado lo mínimo para comprender el ejemplo. Se han pasado por alto muchas cosas que tendrán cumplida respuesta en una próxima entrega.

Para este ejemplo se ha utilizado el compilador de ayudas de Microsoft HC31.EXE, el procesador de textos Microsoft Word 6 y el Borland C++ 4.5. Servirá también el compilador Symantec C++ que se entregó con el número 4 de la revista.

NOTAS

Se ha evitado utilizar términos en inglés. En los manuales originales de las SDK de Windows, e incluso en algunos textos en Castellano, podrá encontrar que se hace referencia a temas como topics, identificador como Context-String, hiperenlaces o referencias cruzadas como cross reference.

CORREO DEL LECTOR

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relaccionado con la programación.

P *Tengo un pequeño problema en el manejo de punteros a grandes zonas de memoria bajo el lenguaje C. Solicito memoria mediante la función farmalloc y la cargo en un puntero del tipo unsigned char far *. Hasta aquí bien, pues claramente me devuelve un puntero a memoria, el problema surge al utilizarla y sobrepasar los 64 Kb iniciales. Aparentemente no hay disponible más memoria y no se porqué. Un saludo.*

Enrique Cartagena Martínez
(Elche / Alicante)

R Su problema es habitual entre la gente que empieza a "trastear" un poco e intenta saltarse la "omnipresencia" del compilador. La función le ha reservado correctamente la memoria, y si utiliza punteros del tipo far (o genera código large) no tendrá mayor problema, y el compilador introducirá el código necesario para cambiar de segmento cuando se llegue al final del activo, con lo cual no hay ni que pensar en la dichosa segmentación, y aparentemente se dispondrá de memoria lineal. Esto tiene un problema: toda la aritmética de punteros far (incluso el apuntar a la siguiente dirección de memoria) es muy lenta, pues tiene que ir comprobando que no se acaba el segmento actual. Lo que se suele hacer es manejar zonas de memoria o datos de menos de 64 Kb, y utilizar punteros near, con lo que se trabaja sólo con el offset, y cualquier cálculo de direcciones se realiza sólo con sumar o restar en él. Aparentemente su problema reside en que, o bien maneja el puntero directamente sin tener en cuenta si se sale o no del segmento, o está utilizando un puntero del tipo near con lo que el compilador ignora cualquier comprobación del límite del segmento.

P *Me interesaría mucho (y creo que a muchos lectores también) que publicaran un artículo sobre programación en el entorno Cliente / Servidor, sus conceptos y sus ventajas. Muchas gracias por todo.*

José de Arévalo
(Madrid)

R Llevamos tiempo en la redacción pensando sobre ese tema, apuntamos su idea y a ver si desarrollamos ese concepto.

P *Tengo varías dudas y cómo escribo de tan lejos, pues hay van todas mis preguntas:*
- Me gustaría conocer los formatos de los ficheros de sonido del tipo VOC y WAV, fuentes para manejarlos y posibles efectos a realizar con los sonidos.
- Para conectar ordenadores (PC) bajo entornos Unix o Linux, ¿es necesario utilizar tarjetas de red, o basta con cables conectados a los puertos serie de ambos?.
- ¿Existen emuladores de memoria RAM en disco rígido?
- ¿Hay aceleradores de lectores de CD-ROM mediante software?
Espero que no sean muchas preguntas y muchas gracias.

Cristian Oriolani
(Mendoza - Argentina)

R Es un placer el contestar a amigos del otro lado del océano y aunque sabemos que los números llegan con retraso, parece ser que no por ello dejan de leernos. Vamos por partes: Sobre los formatos y fuentes sobre ficheros de sonido, ya está realizada una serie de artículos y en breve se publicará. Para conectar dos (o varios) ordenadores bajo un entorno Unix, no es necesario utilizar una red local, basta (y es más sencillo al no necesitar software adicional) con usar cables estándar para conexión serie, que consisten en unos pocos hilos conectados (se publicó hace unos números el esquema de cableado). Cuando se refiere a emuladores de memoria RAM en disco duro (cómo se denomina por aquí) debe indicar sistemas de memoria virtual, tal como la utilizan los sistemas operativos modernos (OS/2 o Unix), y existen librerías y pequeños paquetes que facilitan el disponer de memoria casi ilimitada de manera transparente para el usuario y para el programador. Por último, los únicos aceleradores de CD-ROM que conozco son los drivers de memoria cache que aceleran cualquier acceso al guardar los últimos datos leídos en memoria RAM, incluida la información sobre los directorios y ficheros contenidos en el CD, haciendo casi instantáneo el acceso al disco digital.

P *Quisiera ejecutar un programa residente en un ordenador remoto desde un programa propio (algo similar a la orden spawn de C). ¿Es esto posible?, ¿existen librerías para ello? Muchas gracias de antemano.*

Jesús María Fuentes Navarro
(Toledo)

R Realizar esa operación desarrollando un software propio es un trabajo enorme y, hoy en día, un poco descabellado. Lo mejor sería el utilizar cualquier paquete de comunicaciones existente que permita ejecutar programas y tareas llamando remotamente desde otro ordenador. Uno de los más potentes (y sencillo) es el Terminate, que regalamos en su versión freeware en esta revista hace unos meses.

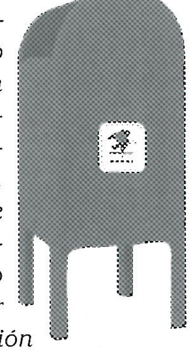
Terminate dispone de un modo host en el que permanece esperando a llamadas exteriores (en realidad se trata de una sencilla BBS) y según el usuario entrante, pues da ciertos derechos y posibilidades, como la de ejecutar programas o procesos batch mediante un menú de opciones.

Con esto se ahorra todo el proceso del manejo del modem, del control de la conexión y del mantenimiento de errores en la transferencia.

Si los ordenadores a conectar estuvieran ya unidos por un cable serie, entonces no sería necesario ni Terminate, pues el mismo MS-DOS tiene pequeñas utilidades para un control limitado de ordenadores remotos.

P *Hola, habitualmente desarrollo productos multimedia y suelo utilizar herramientas y utilidades diseñadas para ello. La calidad y terminación de los resultados es muy aceptable pero penalizan en velocidad de ejecución, y son necesarios ordenadores muy potentes para una visualización correcta.*

He observado que otros trabajos multimedia se ejecutan mucho más rápidamente, e investigando descubrí que estaban realizados directamente en lenguajes de programación como C. Estos últimos, aunque funcionan mejor tienen un aspecto gráfico más estándar y similar a cualquier aplicación Windows. ¿Cuál es su opinión sobre estas dos maneras de desarrollo?



Joan Segarra Casanovas
(Barcelona)

R La eterna pregunta: realizar las cosas mediante herramientas de desarrollo que dan todo hecho, o programarlo todo "a pelo". Realizar las cosas en Clipper o trabajárselas en ensamblador. Utilizar las librerías gráficas que acompañan a los compiladores o desarrollarlas optimizadas a nuestra aplicación, sistema operativo y máquina. Está claro que cada sistema tiene sus ventajas y sus inconvenientes. Si se va a crear una base de datos para los CD en una tarde, el lenguaje Clipper se perfila como la solución más coherente; pero se desea desarrollar una "demo" para participar en el ASSEMBLY '95, o se programa en ensamblador o se muere uno de hambre.

Realizar una presentación multimedia en un sistema operativo gráfico a bajo nivel (programando directamente en C, ni lo intentamos en ensamblador), permite un mayor control de los recursos y unas mayores prestaciones, también permitiría un mejor aspecto gráfico y originalidad, pero como habría que programarse el manejo de cada pequeño botón o control de aspecto diferente a lo usual en el entorno, al final no se realiza por exceso de trabajo, y se recurre a los widgets estándares del sistema.

Otra cosa es utilizar herramientas pensadas para crear dichas presentaciones que con sólo arrastrar un icono a la ventana a modificar nos cambia totalmente su aspecto (realizar esto programando serían horas de trabajo), y contienen cientos de iconos y gráficos para enriquecer nuestra obra. Sus problemas: sus menores prestaciones, y la menor flexibilidad en el tipo de aplicación a realizar.

La conclusión a todo esto, es que se debe elegir en cada caso, en función del tiempo disponible, de la clase de presentación a crear y de los conocimientos del desarrollador.

A menudo nuestros lectores nos preguntan cómo se pueden conseguir los números atrasados, o cómo solicitar las fichas de programación publicadas conjuntamente. Pues bien, aquí se detalla la forma de realizarlo:

Por correo a la siguiente dirección:

TOWER
COMMUNICATIONS SRL
C/ Marqués de Portugalete,
10 - Bajo
28027 Madrid

Por fax al siguiente número:
(91) 320 60 72

Llamando al siguiente teléfono:
(91) 741 26 62

Horario de 9 a 14 y de 15:30 a 18:30 horas.

También estamos recibiendo muchas cartas y llamadas de lectores en la redacción de Sólo Programadores interesándose por diferentes cuestiones sobre el concurso de demos:

El plazo máximo de entrega de trabajos es el 15 de Diciembre.

Se pueden presentar programas con los que también se vaya a concursar en otros certámenes.

Es imprescindible adjuntar la tarjeta de participación que aparece en cada número de la revista.


DESDE SIEMPRE TODOS HEMOS DESEADO
MIRAR A TRAVÉS DE WINDOWS

Usted ya sabe cómo desea su aplicación.

Además, sabe hacer "clic" con el ratón y

trazar una línea.

Ahora puede ver cómo el ordenador hace el resto del trabajo.

Y así termina su aprendizaje en la programación orientada a objetos con VisualAge.

Nadie discute los beneficios de la programación orientada a objetos. La cuestión es si merece la pena el tiempo y el dinero que costaría implementarla.

Con VisualAge™ de IBM, esta cuestión es irrelevante. Su sencillez salva las barreras entre usted y el rápido desarrollo de las aplicaciones orientadas a objetos.

VisualAge está a años luz de los simples constructores de GUI. Es un entorno gráfico que le guía a través de todo el proceso, desde el diseño de la interfaz hasta la ejecución de la aplicación. Como decía la revista *InfoWorld*, "una obra maestra de la programación visual".

Con la versión C++, usted puede trabajar con "partes" de la Librería de clases IBM Open Class, creando vínculos entre ellas. Son fáciles de modificar y cumplen los estándares, de manera que puede utilizarlas en diferentes entornos, (OS/2 Versión 2.11, OS/2 Warp, AIX, Sun Solaris, MVS, OS/400 y en el futuro Windows NT y Windows '95), desde PCs hasta los más grandes servidores.

Cuando su proyecto esté terminado, habrá creado una aplicación con el código más avanzado (C++ ó Smalltalk), y en una fracción del tradicional tiempo de desarrollo, estará listo para implementar una auténtica aplicación orientada a objetos, con componentes sólidos que se pueden reutilizar una y otra vez en futuros proyectos.

Naturalmente, su solución orientada a objetos completa requiere algo más. Por eso IBM ofrece la más amplia gama de productos en programación orientada a objetos, más experiencia y servicios que ningún otro fabricante de software.

¿Puede su software llegar a tanto?

Si desea solicitar más información, llámenos al 900 100 400 (9 a 18 h. de Lunes a Viernes).

IBM Internet: http://www.software.ibm.com



Soluciones para nuestro pequeño mundo